HZ BOOKS
华章教育

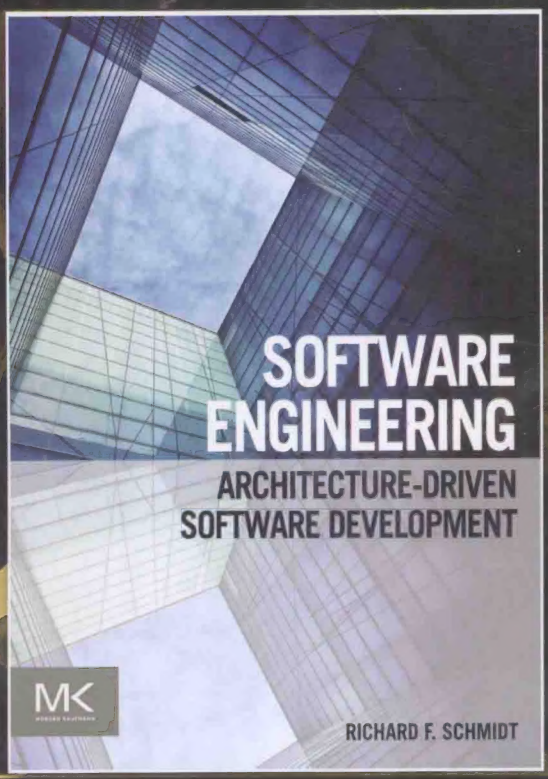
计 算 机 科 学 丛 书

ELSEVIER
爱思唯尔

软件工程
架构驱动的软件开发

[美] 理查德 F. 施密特（Richard F. Schmidt）著
江贺 李必信 周颖 等译

Software Engineering
Architecture-Driven Software Development



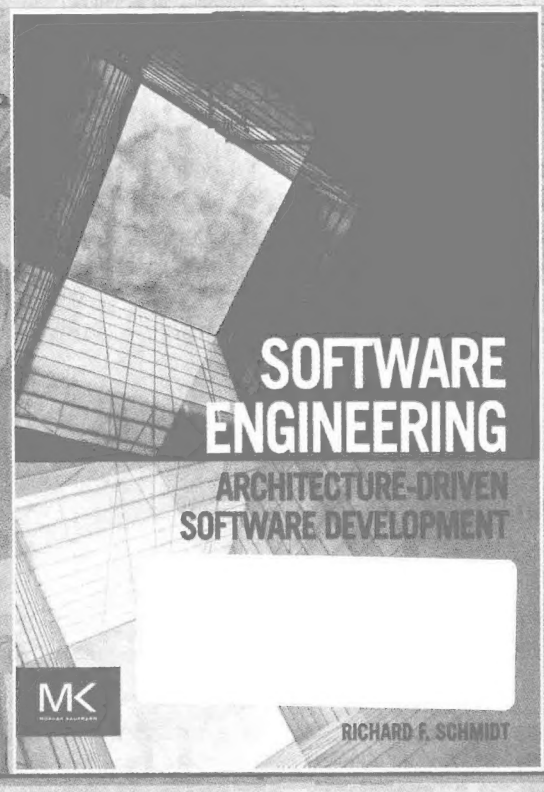
机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

软件工程
架构驱动的软件开发

[美] 理查德 F. 施密特（Richard F. Schmidt） 著
江贺 李必信 周颖 等译

Software Engineering
Architecture-Driven Software Development



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件工程：架构驱动的软件开发 /（美）理查德 F. 施密特（Richard F. Schmidt）著；江贺等译 . —北京：机械工业出版社，2016.7
（计算机科学丛书）
书名原文：Software Engineering：Architecture-Driven Software Development

ISBN 978-7-111-53314-6

I. 软… II. ① 理… ② 江… III. 软件工程 IV. TP311.5

中国版本图书馆 CIP 数据核字（2016）第 140294 号

本书版权登记号：图字：01-2013-7842

本书比较全面地介绍软件工程学科，展示软件工程原则与基于系统工程的软件实践，阐明与软件工程所用的严格方法相关的实践活动、原则、任务和工件。本书共分三部分：第一部分讨论在软件工程体系下的软件开发框架和项目构建；第二部分通过 6 项技术惯例传达一种理念——利用计算技术，应用科学原则以及激活设计软件产品结构的灵活性；第三部分讨论软件工程团队在软件开发项目中扮演的角色，以便建立和控制软件产品架构。

本书适合作为高等院校软件工程及相关课程的教材，也可作为软件开发人员和软件技术人员的参考书。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：谢晓芳　　责任校对：殷　虹
印　　刷：北京瑞德印刷有限公司　　版　　次：2016 年 7 月第 1 版第 1 次印刷
开　　本：185mm × 260mm　1/16　　印　　张：14.75
书　　号：ISBN 978-7-111-53314-6　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brian W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章科技图书出版中心

译 者 序

Software Engineering: Architecture-Driven Software Development

自从1968年提出“软件工程”这一术语以来，研究软件工程的专家、学者们陆续提出了100多条关于软件工程的准则或信条。然而，软件工程项目失败、进度落后、预算超支等问题仍屡见不鲜。部分原因是目前的教育更关注于编程以及模块级的设计，而缺乏对软件架构设计的理解。没有一个完整的设计概念，编码就会很低效，进而造成软件产品生命周期中的设计基础不稳定。软件行业项目成功率不理想正是作者Richard F. Schmidt出版并推广本书的动机所在，他希望借此对未来的软件工程师有所启发。

Richard F. Schmidt在航空航天系统工程和软件工程方面拥有30多年的研发经验。他曾参与防御系统软件开发的美国国防部标准2167A（DoD-STD-2167A）的修订和防御系统软件质量保证的美国国防部标准2168（DoD-STD-2168）的制定。Richard还负责了IEEE 1220标准（系统工程过程的应用和管理）的发布。本书阐述了那些开发政府或企业系统项目的公认的软件工程实践。本书介绍的是跨学科的软件开发方法。本书的内容与IEEE计算机协会的软件工程知识体系（SWEBOK）相对应，重点是整合各种软件开发方法和对开发有效且高效的软件产品必不可少的架构化设计实践。

全书分为三个部分，共20章。第一部分主要介绍软件工程基础，包括软件开发框架、软件架构、项目环境和软件集成以及软件设计的阻碍；第二部分侧重于软件工程实践，这涉及软件需求分析、需求管理、功能架构制定、功能分析与分配实践、物理架构配置、软件设计综合实践、软件分析实践、软件验证和确认实践、软件控制实践等；第三部分重点分析软件工程应用的各个阶段，从软件需求定义到软件验收测试。

希望广大读者可以通过本书对架构驱动的软件开发方式多一些了解，对软件工程过程多一些思考。受译者水平所限，书中如有错漏和不当，欢迎大家指正讨论。

江贺

本书提出了几个颇有争议的话题。这些争议性的话题涵盖了“软件工程”的范围，并且是作者出版本书的动机核心。如果软件工程学科很好地建立起来，并且被证明取得了成功的结果，那么就不再需要出版和推广本书了。然而，并非如此。在过去 20 年中，软件行业项目的成功率都徘徊在 30% 左右。这些项目的失败都可以和两个几乎在软件开发项目或方法中随处可见的主要问题联系在一起。第一个问题涉及对于什么是软件产品设计以及如何开发一个完整的设计描述几乎完全的误解。第二个问题涉及缺乏一套可用于为软件工程学科建立合适范围的软件工程原则和实践的标准。

本书提供了一套全面、集成并且紧密耦合的实践。然而，这些内容背离了当下流行的“最佳实践”，因为它们缺乏设计软件的完美方法。其中的一些观点可能有些批判性；当提出一个方法去修复缺陷系统时，批评是不可避免的。这样做的目的是通过建立一套重要的软件工程原则和实践来鼓励软件社区进行广泛交流。

我希望读者能够保留自己关于软件工程中主流概念的观点。不要让这些有争议的话题分散你的注意力。本书为软件产品的设计提供了一个严谨的、严格的方法。全体软件社区是时候采取行动来提升惨淡的业绩了。我希望这本书能让未来几代专业软件工程师受益。

——理查德·F·施密特（Richard F. Schmidt）

本书旨在比较全面地介绍软件工程学科，展示软件工程原则与基于系统工程的软件实践。本书详细地解释了基本的软件工程体系理念，即强调使用严格规范的方法来设计软件产品。为达到此目的，第一部分讨论了在软件工程体系下的软件开发框架和项目构建。第二部分展示了 6 项技术惯例，它们传达了这样一种理念：利用计算技术，应用科学原则以及激活设计软件产品架构（即设计）的灵活性。第三部分讨论了软件工程团队在软件开发项目中扮演的角色，以便建立和控制软件产品架构。典型软件开发项目的每个阶段都会讨论的重点是软件工程团队如何与其他技术和项目相关的团体协作来影响架构设计和软件产品实现。这几部分阐明了与软件工程所用的严格方法相关的实践、原则、任务和工件。

本书的基础概念基于系统工程实践来达到表 1 确定的目标。这些目标通过应用一系列来源于系统工程学科中 50 多年来成功应用于开发复杂系统的原则和实践来实现。它强调完整软件架构的建立，这使得产品的每个元素都要明确，以便制造、组装、集成和测试。将这些实践应用到软件工程领域，为解决表 1 中列出的那些挑战提供了基础。

表 1　软件工程挑战与目标

软件工程挑战	目标
在编码之前必须先做设计	在提高成本效率和进度准确性前弄清楚正在构建什么
	减少产品在设计细节和精度上的复杂性
	成本管理、进度安排和风险控制
交付软件技术数据包	完整的设计图表和软件实现（构建）的说明文档
分配设计配置元素间的需求	软件组件和单元间的需求分解与分配
	需求可追踪性
集成产品和过程开发（IPPD）	产品维护性能的并行设计与开发
	生命周期成本控制
准备软件集成策略	架构设计活动中计划的软件组件集成开发
	高效的软件实现规划
控制软件复杂性	降低软件维护 / 支持成本
	高效、用户友好的交互
使变更同化	涉众 / 用户满意度
	产品竞争力
权衡分析	成本管理和进度控制
	设计优化
	产品演变 / 增量发布的稳定性
	项目成功率的增强
预先计划的产品提升	将某些功能延迟发布来保证产品按时交付

软件分析与设计的当前实践基于计算机编程语言和这些语言处理数据使用的逻辑概念。这驱动了诸如面向对象设计的软件设计方法，它并不是用来处理先进软件产品的复杂性的。通过适应系统工程实践，本书建立了严格的软件工程原则和实践，从而提供了一种全面的方法来设计软件产品。这些软件工程实践必须清晰说明以保证它们对软件开发的重要性和适用性是确定的。将这些实践应用于一个软件开发过程演练中，以便可以控制、修正和管理贯

穿整个软件开发项目上下文的软件架构。本书的内容与软件工程知识体系[⊖]（SWEBOK）的主要过程领域（如表 2 所示）相对应。与 SWEBOK 的对应说明了本书中的主题是如何根据 SWEBOK 中主题进行安排和关联的。然而，SWEBOK 是基于现在的软件开发实践，严格从技术上来讲，它并不包括系统工程实践。

表 2　SWEBOK 关键过程领域

关键过程领域	本书范围
软件需求知识领域	第一部分：第 3 章
	第二部分：第 7 章、第 9 章
	第三部分：第 17 章
软件设计知识领域	第一部分：第 3 章、第 6 章
	第二部分：第 10 ～ 14 章
	第三部分：第 18 章
软件构造知识领域	第三部分：第 19 章
软件测试知识领域	第三部分：第 19 章、第 20 章
软件维护知识领域	第一部分：第 5 章
	第三部分：第 17 ～ 20 章
软件配置管理知识领域	第二部分：第 9 章、第 16 章
	第三部分：第 20 章，提到配置审核（FCA/PCA）
软件工程管理知识领域	第一部分：第 4 章
	第二部分：第 9 章、第 16 章（提到项目和技术规划，提到工作包）
	第三部分，提到项目和技术方案，提到工作包
软件工程过程知识领域	第二部分
	第三部分
软件工程方法知识领域	第二部分：第 13 章（提到软件设计综合实践面向对象的方法）、第 14 章（提到建模和原型）
软件质量知识领域	第三部分，在测试和评估子部分确定软件质量保证任务

本书内容

接下来会简要介绍书中各章的内容。这三部分将本书内容分成三个连贯的主题，希望读者能增加他们对原则（第一部分）、实践（第二部分）以及软件工程的应用（第三部分）的认识和理解。通过将系统工程实践应用到软件工程领域，本书旨在提供一种创新的、规范的、技术上具有挑战性的方法来开发软件产品。

第一部分：软件工程基础

这一部分讨论的是软件工程相关的基本原则以及这些原则在软件开发场景中的执行。这些基本的原则、实践和理论用于将软件工程建立成一个专业学科。通过讨论软件产品特点和软件开发策略来强调软件开发项目面对的挑战。作为一个组织实体，软件工程弥补了存在于技术专家和项目管理专家之间的外观和感觉上的显著不同。因此，这一部分陈述了软件工程实践与项目管理责任和其他软件开发角色的集成。

⊖ SWEBOK由IEEE计算机学会制定，具体内容参见*http://www.computer.org/portal/web/swebok*。

第 1 章概述了软件工程概念、原则和实践，它是解决设计、开发复杂软件产品面临的挑战所必需的。通过调查软件工程实践和工具来确定它们与项目管理机制的关系。

第 2 章讨论了那些描述软件产品如何定义、设计和实现的软件开发活动的进度。本章通过一系列被项目里程碑和评审分离的连续开发阶段，来追踪典型的软件开发工作，还陈述了软件技术与项目管理控制领域的关系。

第 3 章确定了软件架构的组成，包括以下几个方面：软件产品、计算环境和那些能满足产品维护的开发后的过程。它涉及架构设计表示、模型和文档对技术和项目相关机制和必要性，以保证软件开发工程对在预算内按期交付是必不可少的。需要讨论建立软件需求规格的技术，功能和物理架构要与软件开发阶段一致。本章讨论了软件产品架构如何为软件实现（编程设计、编码、集成和测试）提供结构化基础以及产品生命周期支持。

第 4 章使读者了解那些导致软件开发变得复杂并且难以理解的软件产品特征。它解决了软件产品复杂性挑战，并且将这些与那些已被证明有利于成功完成软件开发工作的项目构建和实践相关联。其中的真知灼见将帮助减少项目的障碍、变动、作废和失败。

第 5 章提出了 IPPD 理论和它对于项目范围的影响以及开发后过程的考虑。它试图证明需要严密构思和结构化的软件架构来确保可以延长产品的使用寿命，这是由于在开发过程中软件工程关注了生命周期问题。同时检查软件开发后过程的工程可以发现，早期的架构决策可以影响生命周期和拥有成本。

第 6 章检查了导致“设计”实践变得非常规并且更加难以理解的软件底层特点。作为挑战传统工程审查的设计和构建材料，讨论软件特征。本章提出了管理软件产品设计的软件工程原则。最后，本章介绍了软件设计歧义来计划一项允许对软件产品进行设计的决议。

第二部分：软件工程实践

这部分确定了 6 个有助于专业化软件工程的实践：1）软件需求分析，2）功能分析与分配，3）软件设计综合，4）软件分析，5）软件验证和确认，6）软件控制。每个实践都由一定数量的任务来表示，每个软件工程的专业人员都应该理解。这些实践建立了一套清晰的任务，集中于设计和细化软件产品架构。

第 7 章提出了一种方法来开发源自涉众需求和期望，以及有助于决定软件开发工作范围的软件需求规约。软件规约驱使软件架构的定义，但不应该推断出任何架构设计方案。软件需求作为派生软件功能和物理架构的起点。架构设计是由表示功能架构和配置物理架而完成的。架构中的每个元素都必须能在软件规约中指定和追踪。要检查软件需求、软件工程任务，以及项目和技术计划之间的关系。

第 8 章确定了必须选择性地应用特定任务来建立软件产品和开发后的过程规约。这种实践涉及软件架构中低级功能和结构元素之间性能限额的分配。这种实践从征求涉众需求和期望的工作开始，以建立软件产品需求基线结束。

第 9 章讨论了以积极主动的方式控制软件架构的重要性，这有利于对已提议的变更进行评估。通过考虑软件需求管理工具和实践以使软件工程团队可以实际考核变更对软件架构的影响和项目资源的自由度来适应一项期望的变更。意图是使开发团队有能力机智地回应得到授权的变更，并且在不扰乱项目范围、计划和成功结束的进度情况下将改进融合到软件架构。

第 10 章讨论了功能架构的本质以及它如何将明确的需求分解为连续的功能元素。每个功能元素在不断改进的方法中是明确的，这些改进会在识别出未完成的功能并且保证它们能实现时告终。功能架构提供了有逻辑性的、连贯的软件产品行为表示法，以回应那些计算环境内出现的刺激、事件和状况。

第 11 章确定了为确保得到完整、一致和可追踪的功能架构而必须考虑的具体任务。通过分析理解业务和软件产品行为，方式包括检查、分解、分类和指定那些来源于需求规约的顶级功能。在功能间分配性能需求来确立低级功能元素有效性和性能的度量标准。

第 12 章描述了安排和确定软件产品物理架构的目的和策略。该物理架构确定了软件单元设计、编码和测试的基本构建快。制定软件集成策略来确定产品结构并规定软件单元和组件如何增量地组合、集成和测试，进而形成完整的软件产品。

第 13 章确定了必须考虑的特定任务，以确保生成完整、一致并且可追踪的物理架构。为了从纯粹的产品功能表示向物理架构过渡，设计综合是一个经过验证的系统工程实践。它涉及“制造或者购买”的权衡，这对应于软件“实现或再利用”决策。

第 14 章确定了必须执行的特定任务，进行设计备选方案的权衡分析和风险评估。进行架构设计决策必须在抑制应用复杂性和软件生命周期成本上有足够的洞察力。与进行权衡分析和风险评估相关的任务可以为理解架构设计决策的本质提供基础，并且对软件开发工作产生影响。

第 15 章确定了必须执行的特定任务，以确保软件架构元素保持一致，并且与授权的变更提议和请求一致。必须执行验证任务以确保软件实现、测试和评估工作与软件架构规约以及设计文档是同步的。

第 16 章确定了选择性应用的特定任务，以确保软件产品架构反映了当前设计思想并包含授权的变更提议、请求和设计决策。需求追踪必须嵌入到软件架构中去，并且与文档关联，这样技术团队才能迅速并且有效地响应变更控制委员会的决策。此外，将授权的变更提议和请求反映在项目和技术计划、进度、预算以及工作包描述中是十分必要的。

第三部分：软件工程应用阶段

这部分讨论了在软件开发项目中，分配给技术组织的角色和职责。强调技术组织在软件工程集成产品团队（IPT）中的参与。

第 17 章确定了由软件工程 IPT 生成的软件需求规约的方式。将参与的组织代表的贡献确定为软件产品的需求，并且建立了开发后的过程。

第 18 章确定了在概要和详细架构阶段定义的软件功能和物理架构的方式。这些阶段关注 IPPD 方法来促进软件实现、测试和开发后的过程必要的基础设施的建立，以推动项目目标的实现。

第 19 章确定了软件实现组织要执行的任务，以编程方式来设计、编码和测试软件单元，并且进行软件集成和测试。在这个阶段同时实现开发后的过程，来支持验收测试和部署准备评审。

第 20 章确定了在软件产品验收测试过程中软件测试和评估组织要执行的任务。参与的组织代表的职责在于监督验收测试、应对测试失败，并且回应由验收测试导致的软件问题报告。此外，开发后的过程必须有资格确认它们已经做好准备来支持软件产品发布、培训和维护业务。

第二部分 软件工程实践

第一部分

Software Engineering: Architecture-Driven Software Development

软件工程基础

本部分概述了软件工程学科，以此使读者熟悉用来描述软件工程原理、实践和任务的词汇。从根本上讲，软件是一种制作软件产品的独特材料。软件作为制作材料的特色对软件专业人士而言是一个谜。通过研究与软件产品工程和设计相关的挑战来减少各种软件工程方法的混乱。通过建立软件工程的基本理论来提供一系列创建软件工程学科的原则和实践。

首先，本部分介绍一组术语，用来讨论健全的软件工程技术的应用。其中的术语汇聚了各个专业，尤其是系统工程、项目管理和配置管理中公认的词汇。这些术语旨在抵制一些软
1 件社区中使用的不合法术语，因为它们会造成混淆，增加建立标准实践的难度。对软件行业而言，至关重要的是稳定该行业同其他工程专业人士、涉众和消费者讨论该行业专业实践的方式。在尝试阐明技术挑战或设计独创性的优点时，出现方言是很不应该的。

Fred Brook（布鲁克斯）教授通过讨论巴别塔阐述了多个软件词典引发的混淆。他的书《人月神话》（*The Mythical Man-Month: Essays on Software Engineering*[1]）确认爆发的泥潭是软件方言泛滥的结果。自首次出版之后，该软件文化便在最短的时间内传播了更多编程语言、技术和方法，比人类历史上任何其他行业都快。这个趋势继续分散、弥漫，进而增加了传统工程行业与软件工匠和软件工程印象派之间的沟通障碍。软件工程是一项要求苛刻的职业，它不允许散漫的技术和方法。软件术语和方言必须基于已经建立的工程方法合并为一个统一的词典。源于多种软件语言和方法的泥潭预示着那些尝试与层出不穷的新颖软件战略同步的产业和学术社区有被淹没的威胁。

因此，本部分会耐心地探讨软件产品开发的各种影响。软件开发工作必须借助这些影响或者环境变量才能成功。因此，本部分讨论被软件产品工程的任何方法接受的各种技术和商业动机。从根本上讲，意识到软件产品旨在对采用它们并提供这一独特产品的企业的经济繁荣做出贡献这一点是很必要的。

软件产品的一大类旨在通过自动化常规的、劳动密集的任务来促进企业多产。这些软件产品降低了企业对人工数据收集、分析、操作的依赖，尤其是在计算错误可能导致减少企业盈利的昂贵错误方面。开发软件产品的公司想要从软件产品开发的投资中享受到有益的补偿。因此，软件开发者和使用软件的企业双方都有兴趣确保软件开发项目得到一个专业的、用户友好的、能正常工作的产品。然而，根据 Standish Group 这 20 年的 Chaos 报告[2]，软件开发项目的成功率却徘徊在 25% ～ 30%。显然，企业正在对软件行业按时在预算内成功交
2 付产品的能力失去信心。失败！混乱！但这并没有困扰软件专业人员，使他们的声誉太过狼狈不堪。

别怕，因为当绝望的专家跳上另一辆开往某处的花车时，总存在另一个自称软件专家的人具有新奇的软件开发解决方案。另一种方法、扣人心弦的新术语和另一套纷繁的活动准备上场。软件开发产业正拼命地寻找一个解决方案，可以显著地为其辩护增加可信度。软件专家无法接受其他工程学科支持的基本原则和实践，因为软件不同于其他产品。千真万确！然而，可能会从其他工程实践中获得一些适用于软件产品工程的价值。

本部分包括 6 个介绍性章节，它们确定了检查软件工程原则和实践的各个阶段。这些内容基于作者过去 20 年的系统工程实践应用经验。1989 年，题为“程序中的漏洞”（Bugs in the Program[3]）的美国国会报告激起了作者对系统工程的研究。这份报告中的发现和建议如下：

---

[1] Brooks, F. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley.

[2] 参见 *http://blog.standishgroup.com/pmresearch*.

[3] Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation, Staff Study by the Subcommittee on Investigations and Oversight, Congress, Sept. 1989.

美国国家科学基金应该确保计算机科学或软件工程课程应该让学生在教育过程中掌握系统工程概念。

政府现在获取软件的系统无法满足政府的需要并且浪费资源。需要“系统工程”学科来弥补采购系统的缺陷……软件开发是一项复杂的过程，需要先进的“系统工程”技术。

系统工程学科推广了一系列处理复杂产品开发活动的实践。与此同时，这些实践已经不当地适应了软件产品的独特本质。第二部分介绍了系统工程相关的6种实践。然而，它们已经适应了软件开发的挑战。

系统工程原则和实践

系统工程是一门融合多学科的技术和项目管理实践来开发与复杂产品相关的架构设计挑战的学科。在考虑整个产品生命周期的同时，它通过对产品业务环境的关注界定了问题域。典型的系统生命周期包括开发、测试、制造、分配、训练、操作、维护和弃置。系统工程的 3
原则包括：

1）系统代表复杂的、人造的产品，其中需要软硬件和人类操作者才能有效工作。必须了解系统或产品在其生命周期过程中的完整设置，这影响到潜在设计方案的可行性。

2）产品是组成复杂整体的相关联部分的组合。

3）产品是人类为了一个特定的（有时是一般的）目标而制造（设计、制造、测试、操作和维护）的。这排除了那些出于兴趣考虑的“自然的”或“生物的”系统。

4）产品在运作上的有效性是应用系统思维的结果，系统思维试图了解各部分如何在一个整体内相互影响、合作。这使得了解业务环境对产品及其组成部分的性能的影响十分必要。

5）产品需要对那些较小的、简单的组件和部分进行层次化安排。若不将问题空间分解成可解决问题，则无法得到复杂产品的设计，可解决问题可以区分一或多个技术解决方案。

6）系统架构代表了一套完整的产品生命周期需求以及产品功能配置和物理配置，这些配置提供了其技术描述。

7）当单个部分或组件制造（或获得）、组装、集成并测试后，产品才算实现。

这些系统工程原则可以轻松地适应软件产品。计算环境提供了硬件组件，代表运作环境的主要元素。软件产品是软件工程工作的焦点，它必须能被终端用户使用。这套软件生命周期流程会稍稍不同，因为软件不是加工得到的，它复制、分配和训练的方法明显地区别于基于系统且涉及硬件的产品。

系统工程相关的实践于20世纪40年代早期建立，已经证实可以解决大而复杂的系统或产品设计挑战。这些实践包括：

- 需求分析
- 功能分析和分配
- 设计综合
- 系统分析
- 验证和确认
- 控制

这6项实践及其对软件产品工程应用的适用性是本书的主要内容。第二部分提供了这些 4
软件工程应用实践的详细定义。为了区别于系统工程实践，这些实践的名称做了细微的调整。这使得承认它们应用中的不同以便解决不同的设计问题时可以应用相同的实践。

总结

这部分的章节提供了建立软件工程原则和实践依据的框架。谨慎地选择表现软件工程原则和实践的词汇的避免混淆。然而，软件工程行业中已存在的语言泥潭以及其他工程原则中的潜在冲突可能有损那些用来表达软件工程概念的精确术语的适用性。为了理解本书，作为表达潜在哲理的一种方法接受本书中的这些术语。它们一旦达到其用途，就可能会被丢弃，并被更合适的单词和短语所代替。

软件开发应该考虑到项目的前后情况。这在设计、实现和测试软件产品的技术工作与旨在确保项目费用、资源和项目里程碑得以满足的管理和控制机制之间建立关联。这强调了一个事实，即所有产品必须设计制造和支持整个生命周期来满足既定的成本和时间计划目标。像 design-to-cost、design-to-support 和 life-cycle cost 这样的术语意味着产品有其感知价值，产品工程必须确保其价值，如购买价格、业务成本等表述的一样，同它为消费者和涉众带来的收益等价。因此，架构设计决策必须考虑到设计解决方案对项目资源和消费者的影响。

架构决策最终对软件开发成本以及产品生命周期中的有效性有着最根本的影响。软件很容易修改，因此涉众坚信通过较低成本可以添加新的特征、功能或改进的用户接口机制。架构决策影响软件产品的结构，该结构促进其修改、扩展和加强的能力。因此，理解软件架构是什么样子、它如何开发、如何通过分析做出正确设计决策至关重要。架构决策为软件产品建立了结构化框架，这使其能对基本变更保持弹性。该弹性使得软件产品在软件开发工作工程及其整个业务生命周期中不断演变发展。

此外，软件工程必须接受集成产品和过程开发（integrated product and process
5 development，IPPD）理论。IPPD 开发表达了产品工程中对软件生命周期并发症的考虑。它促进了集成产品团队的使用，来确保一系列软件技术原则被包含进架构决策制定中。这确保了一个更健壮的结构化框架的建立，以此为产品实现的基础。另外，IPPD 强调开发后的过程和基础设施的并行建立。当产品为部署做好准备时，软件复制、发布、训练和维护过程必须是可用的。

本书的中心主题即架构驱动的开发。这对软件工程范例而言是很基本的，该范例强调了在启动软件实现活动（程序设计、编码、集成和测试）前设计完整软件产品的重要性。软件实现代表软件产品的制造，而软件复制代表软件生产过程。制造是使用原材料将某物制成成品。大规模生产是在大的产业规模下成品的批量制造。软件实现包括程序化设计、编码以及软件单元、模块、常规、对象等的测试。这些软件单元代表着软件产品制造中使用的“原材料”。通过识别并指定这些软件单元来建立物理架构。然后，按照一定方式组装、集成并测试这些结构化单元，以达到完整的软件配置项。这种方法执行的是系统工程中国际标准[⊖]使用的 FAIT（构造、组装、集成和测试）惯例。

最后，本部分介绍了由软件的非物质性带来的软件产品设计的障碍。它通过说明为软件产品的结构化配置定义架构化框架相关的困难介绍了软件设计分歧。它介绍了解决该困境的
6 软件工程原则和实践，并提供了一套严格的、纪律的方法来设计软件架构。

⊖ ISO/IES 26702:2007, 系统工程过程的应用和管理的 IEEE 标准，and IEEE 1220–2005, 系统工程过程的应用和管理的 IEEE 标准。

软件工程简介

本章概述了软件工程作为一个学科，对于软件产品开发的重要性。如本书所述，软件工程使用已证实的实践和源于系统工程学科的技术。这些系统工程方法和工具都已经做过调整，用于应对困扰软件开发行业的挑战。整合后生成的软件工程实践将会提供更严格的方法来进行软件产品的开发。

系统工程人员已经开发出一套原则和实践方法，使得复杂的产品可以在项目框架中进行设计和开发。通过多年的严格探索、反复试验以及从失败中吸取的教训，许多系统工程社区遇到的挑战都已经得以解决。通过采用这些已证实的系统工程实践，要解决的软件开发挑战包括：

1）确立复杂产品的结构。

2）管理与外部系统或产品交互的接口。

3）降低成功项目中的风险。

4）通过考虑设计可选方案和权衡分析来做出明智的决策。

5）以正规的方式来评估变更需求和变更提议，确保采用变更并将变更后要执行的任务维持在既定的预算和时间范围内。

6）平衡不同涉众群体的需求。

7）在产品设计过程中考虑非功能性的挑战和产品生命周期的挑战。

8）在有益组件之间分配性能特征。

系统工程技术用于复杂的项目：航天器设计、计算机芯片设计、机器人技术、软件集成以及桥梁建设等。系统工程使用的工具包括建模与仿真、集成产品团队和调度管理复杂性。阿波罗项目是大型、复杂项目的典型实例，它成功使用了系统工程并且被认为是人类历史上最伟大的科技成就。[⊖]美国宇航局和国防部一起创立了系统工程并为所有主要的系统开发项目提供基本原料。[⊖]

用来解决这些挑战的系统工程实践包含大量技术和项目管理工具。系统工程可以提供技术和项目管理目标之间的交集。这种交叉的重点在于确保产品设计挑战、问题和解决方案通过项目监督控制链恰当地进行交互，以便技术组织发放指令。这涉及将技术性策略、计划和进度表转化和集成到项目级别的计划和里程碑中。这些技术工具和项目管理工具构成了定义软件工程学科的基础。本书将对这些项目管理工具和技术工具进行讨论并检验，然后把它们作为软件工程学科的一种经过训练的方法。主要的技术工具和项目管理工具在表 1-1 中列出。

⊖ 参见 http://history.nasa.gov/ap11ann/introduction.htm.

⊖ 参见 http://spacese.spacegrant.org/uploads/images/Art_and_Sci_of_SE.pdf 和 www.incose.org/secoe/0103/ValueSE-INCOSE04.pdf.

表 1-1　项目管理工具和技术工具

项目管理工具	技术工具
需求规约实践	软件架构
集成产品和过程开发（IPPD）	集成产品团队（IPT）
工作分解结构	工作包
规约树	文档树
集成总体方案和进度安排（IMP/IMS）	集成技术方案和进度安排（ITP/ITS）
项目评审和审核	技术评审和检测
成本效益分析	权衡研究 / 分析
风险管理	技术风险评估
变更提议	变更请求

这些工具已经被纳入软件工程实践，它们用于克服和软件产品设计开发相关的挑战。表 1-2 提供了一个矩阵来确定软件开发过程中的挑战与软件工程实践和工具之间的关系。这些实践方法会在接下来的篇幅中简要验证。

表 1-2　软件工程实践和工具对应的挑战

挑战 / 工具和实践	需求规约 / 软件架构	集成产品和过程开发、集成产品团队	工作分解结构、产品分解结构	规约树、文档树	集成总体规划和进度安排、技术方案和进度安排	项目评审和审核、技术评审	配置管理、变更控制	成本效益分析、权衡研究 / 分析	风险管理、技术风险评估	建模与仿真原型
复杂系统结构	×	×	×	×				×	×	×
外部接口	×	×				×				
降低风险								×	×	
设计可选方案	×					×	×	×	×	×
项目管理集成		×	×	×	×	×	×			
各种涉众	×	×					×	×		
产品生命周期注意事项	×	×	×		×	×				
产品性能	×							×		×

8 ~ 9

1.1　明确软件需求

为软件产品建立需求是一个极其重要的任务，它能指导后续软件开发的工作方式。传统上，需求规约解释了整个产品的开发流程及其外部接口。然而，用于大多数工程学科中的一个重要实践工作是产品架构或设计中每个元素的需求规约。因此，这种有很大影响力的实践工作需要一个规划完整的软件架构，包括软件产品中每个元素的规约说明和开发后的相关维护流程。

产品的软件需求规约指导了产品架构、软件实现和软件测试的定义以及评估等工作。某些不必要的、过于明确的或者会引入不可接受风险的需求将会给项目带来失败的风险。这说明会出现这样一种情况，软件开发团队可能会尝试在某些方面做较多工作，而在另一些方面做较少工作。项目受限于可用来生产产品的资源的总量。建立产品需求时主要焦点是项目预算和进度目标。一系列复杂的竞争因素阻碍了需求标准的制定，包括：

1）关于软件产品，多个涉众都有各自独立的想法。

2）竞争并渴望得到更大的市场份额。

3）需要建立软件开发的基础设施、环境（架构、实现和测试）、人员配备等来方便实现项目目标。

4）软件开发和开发后的维护流程，要在产品生命周期中同步。

5）企业的目标投资回报率和在本行业中增强的声誉。

每个软件产品都是为一个目标服务的，并且软件需求应当为该服务目标表明产品的功能和性能等因素。软件产品可以支持某些业务流程，控制系统或进程的操作，支持数据收集和分析，通过简单的自动化任务来提高工作效率，或者提供一些相关的娱乐内容。因此，对于每个软件开发工作来说，都存在一种重要的成本效益激励，这是必须要认识到的。建立软件需求时，若拓宽后开发工作的范围超出了实现项目目标的手段，务必要谨慎。不当扩展软件产品范围注定会导致开发工作的失败。每个需求都意味着，要设计出一个合适的解决方案，就必须付出足够多的努力。管理好软件工程任务的范围对于每一个成功的开发项目都是至关重要的。

10

1.2　软件架构

软件产品是由软件例程、程序、模块或提供功能的对象组成的。作为一种开发产品，软件不具有物理特征。软件实际上是一种语言，它可以在允许数学计算的处理单元中转化成电流。软件命令是可译的，这允许数据操作，或者或者更精确的说法，允许了函数，它表示在调用时计算机会产生单个结果的基本操作。因此，软件产品的设计是至关重要的，它必须要满足最终成品能体现出来的所有功能。软件架构理论提出将需求分解成可以提供明确功能和性能特征的函数和子函数。软件架构指的是设计和实现软件产品的艺术和科学。它包含三部分：1）产品需求；2）功能架构，即可以展现出功能、性能和资源利用特性；3）物理架构，即建立软件产品的结构配置和各个结构元素之间的联系。

软件架构类似于工程图纸和施工图。建筑物只有在一系列的设计图起草完成后才能开始建造；建设过程中要遵照统一的建筑、机械和管道规范，还有国家的电力规范；此外还需要授权管理机构的批准。同样，软件产品的实现（模块的设计、编码和测试等）也需要先完成软件架构的设计才能开始，这样可以体现出是否符合软件需求，并授权项目进入开发的实现阶段。没有了解整个工程责任的范围就开始“建设”是不明智的。

软件架构为这些经过严格探讨的、优化的、在既定预算和进度规定范围内实现的软件产品建立了一个完整的设计框架。术语“设计”在百科全书字典中的定义是：“为某些事物的形式或结构做一个详细的计划，强调其特征，比如它的外观、便利性或者高效运转。”这个定义识别 4 个重要特点来确定它是否适用于软件架构理论：

1）为某些事物的形式或结构做一个详细的计划：这里的“计划”意味着要用一套工程图纸来从各个角度描述产品的形式或结构。软件架构提供了几种形式的设计图表和视图来明确软件产品的结构和性能。在与开发团队的成员和其他涉众进行架构概念沟通时，这些视图是十分必要的。

11

2）外观：用户或操作人员可以观察、交流和控制软件操作的机制。典型的用户界面元素包括声音、指示灯、窗体、对话框和报告格式。软件产品的外观对于消费产品来说可能略显随意，如果需要较为先进的人机界面设计可以寻求专家的帮助。

3）便利性：软件产品的可用性，即在控制流程、数据录入、数据检索、数据操作和报

告生成等方面操作简单。

4）高效运行：这个特点解释了设计软件产品的原因。该特点表现在其功能、操作和性能以及计算资源的利用率上。软件架构设计旨在充分利用可用的计算环境资源，例如，处理速度、数据传输速率、内存、数据存储和通信带宽。

显然，一个软件产品的架构的定义和设计需要一种严谨的方法，以及获取和表达架构设计特点的技术。软件工程涉及一些设计上的挑战，包括计算机技术、软件组件、人为因素工程学，以及与其他系统和应用程序的接口问题。软件架构包括：

- 与软件产品涉众交互而派生出的需求规范规约集。
- 在用户、运营商和外部系统之间的软件行为和交互的功能表示。
- 在物理或者结构上安排这些软件“构件块”，并且利用某些策略将这些软件元素组合成单一完整的产品。

1.3　集成产品和过程开发

集成产品和过程开发（IPPD）是一种可以提供系统化产品开发方法的组织结构技术。该技术致力于在产品开发期间，通过相关涉众及时沟通来提高产品质量，更好地满足涉众需求。集成产品和过程开发的基础租户包括开发过程开始时的所有技术学科，以确保可以正常地收集、理解和指定需求。它的目的在于鼓励开发人员考虑产品生命周期的所有方面，确保产品架构对业务和技术方面的变更具有弹性。集成产品和过程开发理论确保软件工程集成产品团队（SWE-IPT）的成员都可以代表所有技术和管理组织。

需求开发首先出现在软件产品层面，然后将需求分解成功下放到下一级，最后贯穿于整
12 个软件产品架构。这与传统的软件开发模式不同，传统模式中需要软件分析师先进行需求定义，然后按照需求进行产品设计、实现、测试和评估，开发后的维护过程。这很容易造成因交流上的不同步而导致的理解偏差。执行集成产品和过程开发技术的一般方法是对于所有产品和开发后的过程，形成多学科团队来解决技术问题，平衡需求并且帮助整合各类团队。技术和管理团队的参与将使产品的整个生命周期发生变化，因为工作会在软件开发的各个阶段发生转变。

1.4　集成产品团队

集成产品团队（IPT）是一种贯穿于整个软件开发工作中，确保涉众、项目管理和技术组织都被代表了的机制。SWE-IPT 涉及的代表来自不同的涉众团体、软件开发技术和管理组织。图 1-1 描述了几个相关的软件开发集成产品团队的组成。推荐的软件开发集成产品团队定义为：

1）软件工程 IPT：该团队主要负责定义和控制软件架构，整合技术方案和进度安排，执行架构交互的研究和分析，以及监督软件开发的进度和风险。

2）软件实现 IPT：该团队主要负责按照实现方案和进度进行软件开发，包括设计、编码和测试软件构件（如软件单元）；集成并测试软件组件；测试集成产品。

3）软件测试和评估 IPT：该团队主要负责软件产品的设计、细化和验收测试。代表客户、项目经理和企业，在模拟和真实操作下，进行软件产品质量、性能和特征的评估。

4）计算环境 IPT：该团队主要负责待开发软件产品在预期计算环境下的计划、定义、

实现和测试。

5）开发后的过程 IPT：该团队主要负责产品复制、分配、测试和维护的计划、定义和实现。

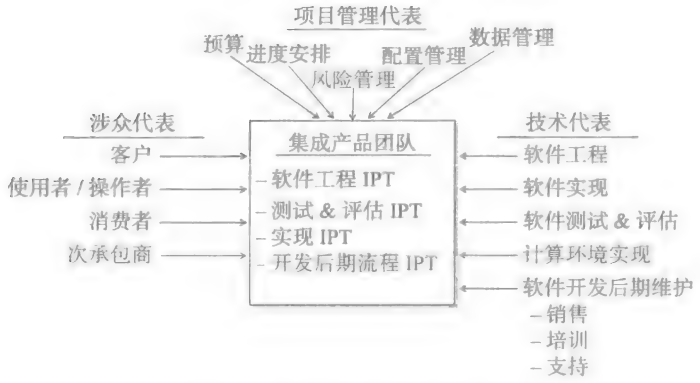


图 1-1　软件集成产品团队代表

一个集成产品团队需要在产品的整个生命周期中负责定义产品和各个开发环节，包括软件架构设计、实现、销售、培训和支持。一个典型的 IPT 一般由软件设计、实现、测试和评估、特定设计（可维护性、安全性、人为因素、后勤保障等）以及客户和管理代表组成。集成产品团队应用先进的方法和工具来制定计划、收集信息、进行权衡分析，以及建模与仿真，这样可以显著提高 IPPD 的有效性。每个 IPT 都应努力从 IPPD[⊖] 中获得以下优势：

1）减少产品交付时间。改变以前顺序决策的方式，根据所有涉众的要求，综合多个角度同时进行决策。所有的决策都应该基于系统软件生命周期的角度，将软件的开发和调度中的量级最小化。减少后续扩展和昂贵的返工周期，这会对软件开发的进度安排和生命周期成本产生积极的影响。

2）降低系统和产品成本。在软件开发过程的开始阶段，适当强调 IPPD 有助于优化产品和各环节的融资状况。基于历史数据的 Pre-IPPD 资金配置可能不再相关。在软件项目的初始阶段可能需要额外的投资，但是由于设计方案的变更较少，单位成本和整个生命周期的成本可能会降低，更好地适应目标进度和权衡分析的广泛使用，生成具有成本效益的解决方案。

3）更好地降低风险。在软件开发的早期，制定团队计划可以促进对可用技术和开发过程的理解。反过来，这也可以对风险，以及风险对于成本、进度和性能的影响有一个更好的认识。有效的风险评估可以在方法和流程上降低潜在风险，并且帮助我们确立更加真实的成本、性能和目标进度。

4）提高产品和开发过程的质量。团队合作和管理可以为软件产品和开发流程的改善提供支持，为相关企业和涉众提高产品和流程的质量。

13 / 14

⊖ 软件收购的黄金实践，集成产品和过程开发（IPPD），http://goldpractice.thedacs.com/practices/ippd/.

1.5　工作分解结构

工作分解结构（WBS）是一种有助于理解软件开发工作和建立工作范围的项目管理工具。WBS 被用于分解那些定义开发工作范围的工作包。每个顶层工作包都涉及多种必须参与执行和控制任务的技术和管理学科。这些工作包被分解为特定的任务、预算分配、交付成果和贡献于某任务的每个组织的里程碑。每个工作包都应确定该任务与构成 WBS 中的其他任务之间的依赖关系。工作报告必须由每个贡献组织通过确定必要参与水平来评估，以确保任务性能满足既定的政策和程序。还要结合组织性任务评估和总结来为该工作的执行提供一个完整的评估。通过积累顶层工作包的工作量、成本和进度安排参数，为确定项目计划、预算和进度安排提供基础。

WBS 中最高几层提供待完成任务的项目框架定义。接下来的几层确定一套完整的技术和待完成的组织性任务。开发过程中进一步进行 WBS 分解，必要时为待完成任务提供详细的技术和组织性任务评估。

伴随着软件架构的建立，WBS 技术势必会逐步发展起来。这就允许了 WBS 来解决技术和相关项目的工作计划，并在开发过程中适当地反映软件架构。软件工程 IPT 负责整合、控制和维护 WBS 项目中的技术元素。这是为了确保 WBS 技术符合软件架构演化的定义和适当地反映待执行的技术工作。

1.6　软件分解结构

WBS 的技术性输入依赖于软件架构，直到关键设计评审（CDR）后才进行定义并不断发展。软件分解结构（SBS）关注的是在需求确定、实现和测试这几个阶段的不同层次的软件元素。现在面临的挑战是软件架构和与之相对的 SBS 方式是随着时间的推移和架构设计方式而变化的。架构方面的决策可能会影响到软件需求以及功能上或者物理上的定义。图 1-2 显示了 SBS 技术伴随着软件架构的定义和细化而逐步演变的过程。对于软件架构的每一项精化、工作定义和资源估算应该变得越来越精确，并且在初始项目预测的范围内越来越容易维护。

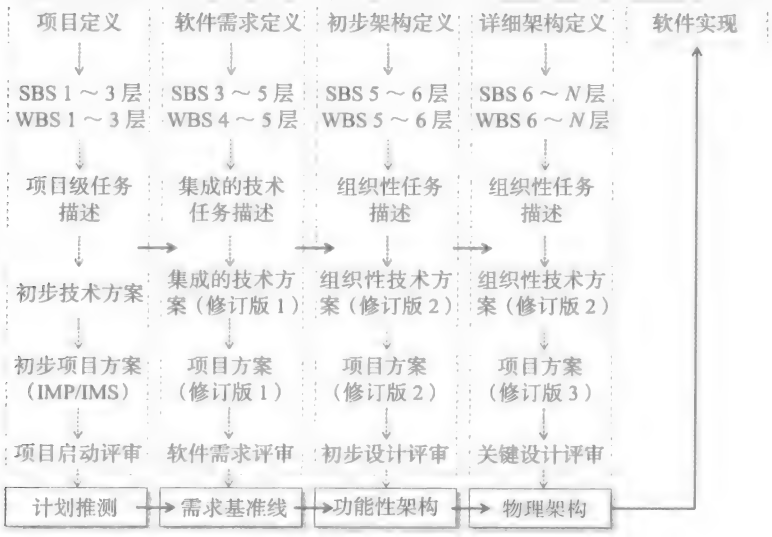


图 1-2　工作分解结构（WBS）的演变

软件开发各阶段的工作流遵循一个典型的分时段的增量式项目序列。在开发的初始阶段解决了产品需求。然后在设计阶段建立了功能架构，确保软件产品必须实现什么来启用每一个操作上的数据处理业务。在第三阶段，细化架构的定义，将功能的表示转换成能够为软件实现提供基本规范和准则的结构化配置，即物理架构。这三个阶段提供了详细的架构信息、规范、图解和草图来保证产品的实现（制造、组装、集成和测试）。在开发的最初的三个阶段，软件分解结构逐步深入（分解的层数）并且更加详细（技术上的准确性）。随着分解的不断深入，软件架构将会不断提供建立 SBS 必需的结构化信息。由于在每个开发阶段，软件架构都能逐步提供更全面、更准确的软件产品描述，这使得接下来的产品实现和测试工作越来越准确。软件架构的定义和发展导致 SBS、WBS 以及技术性和组织性的计划和进度表不断变化。

15 / 16

1.7　规约树和文档树

规约树标识了软件产品和产品配置中每个元素设计和测试所必备的需求规约。对于一种软件产品来说，规约树标识了与每个软件配置项、软件外部接口和计算环境相关的需求规约。除此之外，每个开发后的过程（如复制、销售、培训和支持等）都应该准备一个流程规约。规约树是一种需求规约同最终交付产品一致的项目管理工具。

文档树是一种可以处理为支持软件开发工作、有必要生成的相关规范、设计文档、图表的技术性管理工具。它可以提供一套有关于软件产品和开发后期维护过程所必需的文档集的完整视图。文档树应当包含所有的技术性文档，包括技术方案、软件架构描述、软件开发文件、测试程序、安装指南和用户手册。

SBS 技术、规约树和文档树都是很有用的工具，主要用于理解软件开发工作的实际进展和完成产品准备阶段向操作和维护阶段的过渡。此外，SBS 技术和文档树在软件开发工作中决定变更提议的全面影响和其他影响也是至关重要的。

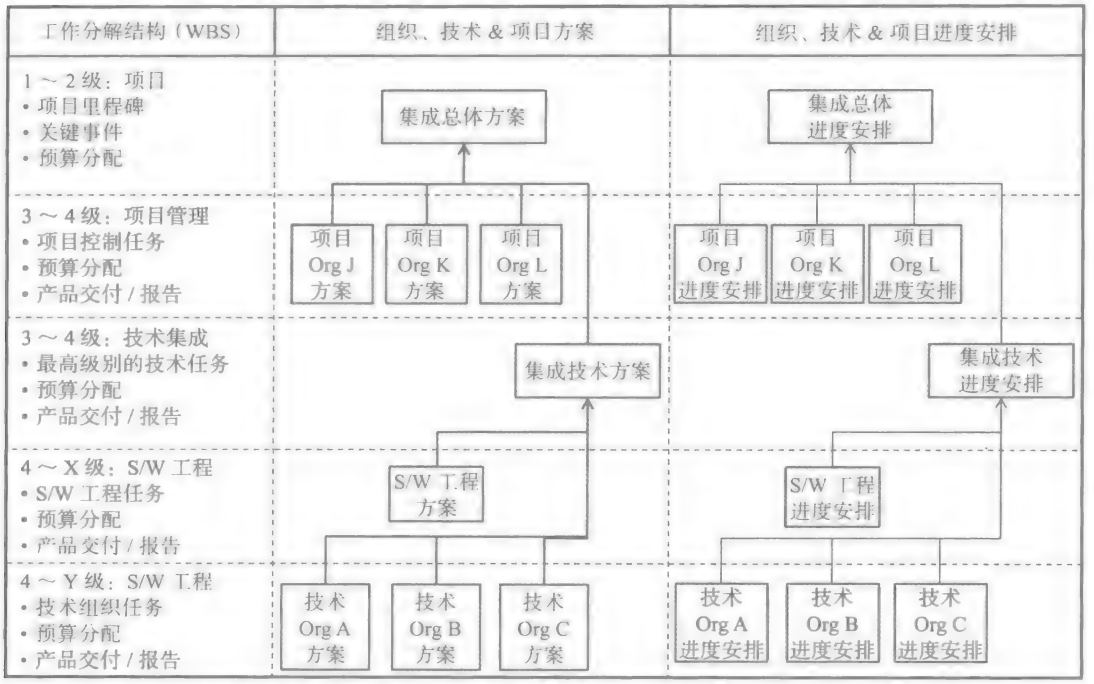
1.8　集成总体方案和进度安排

集成总体方案（IMP）是一个用来记录实现的重要成果和与该成果相关的关键程序事件的事件驱动项目计划。集成总体进度安排（IMS）是一种确保项目成功执行的，包含网络和细化必要任务的时效性进度安排。IMS 对于集成总体规划、WBS 以及任务声明和项目指令来说都是可追踪的。IMS 被用于验证会议计划目标的进展，以及应用组织性集成技术方案来整合计划项目活动。

17

WBS 为建设项目、技术方案和进度表提供了基础。这是各种项目组织准备的方案和进度表层次关系。图 1-3 描绘了一个关于各种组织、技术、项目方案和进度表的一般 WBS 层次关系。随着项目、系统架构和 WBS 在定义和清晰度方面的变化，组织、技术以及项目方案在细节方面可以被扩展。集成技术方案是由集成技术组织方案和系统工程计划结合而发展起来的。IMP 和 IMS 整合并总结了该项目，并且集成了技术方案和进度表。

这些方案和进度表的层次提供了一种关系分离，因此项目管理和技术组织可以在整个项目环境下解决各自的角色和责任。确定项目工作中修改建议的全面影响需要考虑所有的方案和进度表，并且需要修订工作包以便为将变更整合到项目工作流中提供基础。



X——S/W 工程 IPT 定义的最低级别（不包括技术组织细节）

Y——组织细节的最低级别

图 1-3　组织、技术、项目方案和进度表的层次结构

1.9　评审与审核

一个典型的软件开发项目是由一系列时间段来定义的，包括产品定义、设计、实现和测试。每个阶段的活动都以产品开发现状的评审作为结束，并涉及大多数的涉众。进行软件架构、实现和测试状态的技术评审以确保技术组织对项目评审做好准备是必要的。除此之外，IPPD 理论扩大了这些评审的数量和范围来解释软件开发后的过程的定义和状态。图 1-4 提供了一个支持正式项目的技术评审的概念性规划。第三部分为每个正式评审提供了一个详细的过程，包括解释项目和产品的状态、关键架构决策、评审后的提议的变更，以及软件开发下一阶段的方案。

18

技术评审解决了开发中软件产品的问题，而开发后期的评审则解决了开发后的过程的发展。技术评审解释了演化的软件需求以及功能上和物理上的架构的状态。接下来，开发工作会过渡到关注软件实现，包括一系列技术评审来评估软件编程设计、集成和测试活动的合理性。项目和技术评审的主要不同在于步骤和参与的水平。技术评审不涉及项目管理或涉众的参与。

开发后的过程的评审解释了销售、培训的状态以及维护过程的定义和实现。这些过程必须在软件产品部署准备评审前定义、设计、完成、合格，以证明软件产品已经完成，并且准备好投入操作中去或者分发给客户或消费者。部署软件产品的决定必须考虑每一个开发后的过程开发工作的状态。

图 1-4　一系列项目和技术的评审及审核

1.10　配置管理和变更控制

配置管理在产品的整个生命周期中维护了功能、结构和性能特征上的一致性。配置管理实践中涉及配置识别、变更控制、配置状态统计和配置审核。这些实践的目的在于确保软件产品符合其规约、设计、业务功能和支持文档。配置审核确保最终产品的配置经过了正确的测试，所有物理特性都符合规约、图表、图纸和用户手册等，以及确定所有授权的变更都被纳入配置数据（工件）中去。

管理变更是所有开发工作中最重要的方面之一。没有任何开发团队能够预测到软件开发工作中将会遇到的众多问题。术语“开发”有几个含义，包括变得更大、更强或者更令人印象深刻，以及使解决问题或者阐述一种情况变得更容易。这意味着在项目的初始阶段，没有正确理解问题空间来准确定义的一个方案或进度安排，以至于与原始的解释不会有太大变化。当考虑到变更控制时，就必须要承认以下基本事实：

- 建立一个项目是为了开发一种软件产品，并且它被成本、进度目标和资源约束所驱动。
- 项目规划会建立一个路线图来使得项目可以从最初开展并执行下去。随着开发工作中对问题空间和解决备选方案的分析，项目计划和进度制定都必须要根据项目前景的最新了解来进行调整。
- 项目管理涉及众多的控制机制，为了实现建立项目目标，包括预算、资源分配和风险跟踪。
- 软件架构为建立产品结构配置提供了技术框架。这种产品配置将会随着分析、调查、建模和原型设计而派生出更多的细节而发展。

19 / 20

- 随着产品配置被不断细化，产品和 WBS 势必会被扩展，从而为修改技术计划、进度安排和关键里程碑成果标准提供基础。
- 成本和进度目标迫使项目向着其最终成果发展。外部力量，例如客户和涉众的需求和期望、计算技术、竞争和市场条件，都会从确立方案开始就对项目施加压力使其不断发展。

这些断言表明，项目管理和软件工程实践应该以这样一种方式建立，即控制好这些截然相反的力量。开发项目目标驱动的本质是关注以及时并且保证成本效益的方式开发和交付软件产品。这授权了一种设想的策略来隔离项目与变更来源。市场的本质表明，软件开发项目必须认识到开发环境中不断变化的条件将会最终决定软件产品的可接受程度和成败。企业对项目的投资在于提供人员、设施、工具和设备，以及项目实施所必要的资源。由于他们的投资，企业希望了解该软件产品的利润，或者希望通过其可靠的软件开发组织提高其在行业中的知名度。

为了妥善处理变更的动态作用，核心软件工程理论必须支持一种技术性方法来促进以下原则：

1）变更是不可避免的！必须要能区分哪些变更是有利的，可以采纳；而哪些变更应该拒绝或者推迟，直到未来出现其他版本的产品并适用。

2）被采纳的变更必须要能以最小的返工量或进度中断数被纳入计划、预算和产品配置中。

3）每种变更都代表一次返工，除非这种变更是一个全新的需求，不需要与软件架构中任何其他的元素进行交互。然而，即使这个变更是独立的，也需要项目方案和进度表更新来将修正的工作范围纳入工作包中。

4）变更会影响涉及试图操作的软件产品或计算环境的软件架构。应该进行变更分析来理解某个变更引起的感知影响，以支持某个项目决定采用该变更。取决于软件开发工作的状态，涉及纳入变更的返工量将会发生显著的变化。在某些情况下，某种变更可能需要将已完成的工作重新进行。变更的范围可以通过评估受影响的软件架构中元素和接口的数量来确定。这将提供一种作为已有设计和必须返工的工作量的指示。规约树和文档树的评审将会支持变更影响决定。

21

5）在采纳变更之前，必须清楚工作包调整的范围以确保项目成本和进度目标仍然可以达到。软件工程 IPT 必须能确保提议中变更的影响能彻底明确地支持采取该变更的成本效益分析。

6）变更影响分析应该足够详细，以便于能够被软件架构、技术方案和进度表所纳入，并且不给项目的成功引入额外的风险。

对项目变更的期望可能来源于外部，比如客户、竞争或者计算技术的进步，又或者来源于技术团队对于软件架构方案的一种更深刻的理解。这些从内部提出的变更请求提供了一些可以改进解决方案或确定必要的修改方案来解决现有架构缺陷的机会。这些源于架构缺陷的变更应该主导技术变更控制委员会的关注方向，因此这些调整必须要在现有的技术范围内执行。

来自外部的变更表现为短期的变更提案，就与当前工作方案相关的资金和资源而言，是典型的超出当前项目范围的变更。变更提案影响了技术工作方案的范围，务必要项目变更控制委员会的授权来确保所有涉众了解到与该变更授权相关的成本和收益。因此，术语“变更请求”用来指不改变项目资金和进度规划的变更。然而，“变更提议”则是指影响了计划工

作范围的一种改变。在项目考虑某个变更提议之前，该变更提议必须伴随着资金救援或时间救援。

1.11　权衡分析

有许多方法来进行技术、项目和业务分析，从而收集信息来帮助决策。成本效益分析（CBA）是一个系统化的过程，用于计算和评估某个提议操作的成本和效益。这种分析有两个目的：1）确定该操作是否为一个良好可行的操作；2）将该操作与能够实现相同目标的可选方案相比较。这涉及比较预期收益和每个与其相对的替代方法的预期总成本来决定收益是否超过了成本和超过了多少。这种技术对项目级别的决策来说是很有用的，因为它们最终必须以一种金融实效衡量标准来解决。然而，技术决策需要能够处理某些状况的技术，包括设计因素，比如性能或计算资源利用率，以免让自己承担经济后果。

权衡分析提供了一种方法来解决架构问题和应对不确定性决策。在很多情况下，技术方案必须要经过产品性能、可接受性和市场上的成功这些方面的优缺点评估。最终，如果解决方案会影响到项目的范围，那么架构解决方案的实现必须量化成本效益。软件工程涉及项目既定范围内的技术或设计挑战。尽管会出现不确定和不可预知的结果，这样做仍旧是为了进行更好的架构决策。图 1-5 给出了一个关于项目开发的技术和工程领域中权衡分析和成本效益分析使用的图例。 22

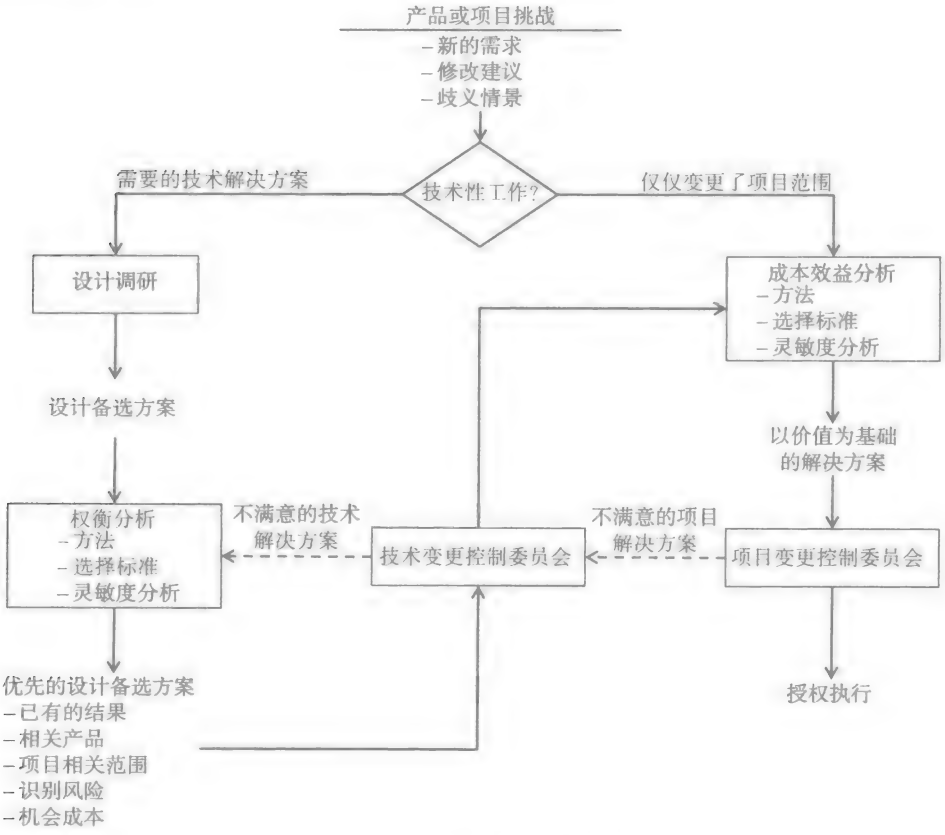


图 1-5　项目和技术决策领域

由于权衡分析用来处理不确定性和某些难以定量的结果，所以不断发展创新性方法来帮助评估设计方案的优缺点。质量功能部署（QFD）是一种流行的技术，已经在全球范围内被广泛应用于汽车和电子行业。质量功能部署利用矩阵来安排，并且在客户的需求（客户的心声）和满足客户需求的产品设计之间建立联系。适当使用质量功能部署可以增加组织中的跨
23 功能集成，尤其在市场营销、工程和软件实现中。质量功能部署分析也可以级联解决低层设计方案，并且允许进行全局分析来提出解决方案，即范围从客户到项目以及产品架构细节的解决方案。

1.12　风险管理

风险管理提供了一种处理已识别并且会威胁到产品或项目可行性的不确定问题的方法。已经被识别的风险会根据它们的严重性（潜在后果）和可能性（发生的概率）来进行量化。已识别的风险会综合考虑严重性和可能性，从而评级为高、中、低，并且被放入风险跟踪框架进行监测和报告。应该找到风险规避的方法，制定一个风险管理计划将各种情况细化，预示出风险的表现形式，并授权激活预防措施。对于风险严重程度高和发生概率大的项目，应该重新审视软件需求或者项目范围，从而找到方法来消除、避免或者降低风险所表现出来的威胁。

未被识别出的风险对于一个项目来说像癌症一样危险，因为随着时间推移它们会慢慢地破坏这个项目或者危及软件产品开发部署的成败。因此，所有软件工程核心元素的权衡分析和架构决策都涉及风险评估。权衡分析涉及架构选择的评估和每个可选方案相关的优缺点。每一个可选方案都必须进行风险评估，以确保潜在风险在执行决策前被识别。

软件开发是一项非常复杂的任务，并且充满了潜在的风险。在对项目的健壮性造成不利影响之前，风险识别包括风险发现、定义、描述、记录和交流。风险识别的一个重要方面是尽可能多地捕获风险。风险识别活动过程中，所有潜在的风险都应该被考虑到。风险识别是一种头脑风暴的形式，而且最好是以无限制或非结构化的方法进行。不是所有的风险都会发生，一旦收集到每种风险的更多信息，就可以针对处理某些不可避免的风险做出决策。这对于搜索领域，针对潜在的煽动性挑战情况下将会发生什么、考虑事件或者架构特点来说是很重要的。

1.13　建模与仿真

许多工程学科利用建模与仿真技术来支持实验设计和分析。计算机技术已经提供了一种简易方法来构建这些模型并且避免了像从前那样使用单调的产品报告作为测试对象。现在的
24 制造业利用复杂的计算机生成模型或者虚拟原型来进行设计评估，以达到显著实验效果和节约成本的目的。然而，虚拟原型中的概念很难恰当地转换到软件开发领域中。

软件行业已经接受了将原型的概念作为一种实现软件产品的实时和增量交付的方法。很多软件开发策略都采用了原型理论。这些方法都慎重地将一种转换计算引入软件开发的方法中。通过对软件产品进行原型设计，软件开发人员可以做他们最擅长的事——编程。等原型已经确定完成的时候，软件开发项目中绝大多数的可用时间和劳动力都已经被用尽了。这种情况掩盖了以下事实的表象，即开发一个最终能被认可的“不断演变的原型”来得到可交付的软件产品。（这么看来，该产品是在这个项目中开发出来的，因此它一定就是最终成品！）

这种做法跳过了运用严格的工程需求来建立一个稳定的架构，以便软件产品可以持续发展。随着原型通过增加额外的功能而不断演变，脆弱的底层架构注定会破裂。这便不是一个合理有效的软件工程实践。

根据定义，原型是建立的一个用来测试概念的或行为的设计实体模型。[⊖]原型设计尝试模拟或模仿了一个设计实体，让工程师或设计师以此来探索设计方案、测试理论以及确定工程预期。原型服务于可以实现的产品规范，而不是基于理论或预期工程解决方案的规范。软件原型从根本上来说是一个矛盾，而且设计这样一个非专业的矛盾是为了使得软件开发人员关注于编码，而不是构建软件产品。

软件原型设计在软件工程中能否达到目的取决于使用的是否合适、谨慎。设计图形用户界面（GUI）就是合理使用软件原型设计的一个例子。GUI 测试报告可以体现出人为测试的主要内容，用于收集可以用来提炼 GUI 规范和设计主题的人机界面数据。进行软件原型设计要谨慎，以确保原型开发过程中的信息收集（原型设计的优势）是有价值的。这是正确的！原型设计是开发过程中的一种形式，并且在被用于收集工程数据或用户反馈之前，软件原型设计必须有合适的范围、说明、设计和实现方式。这表明许多严谨并且细致的、和软件产品实现相关的实践都可能被忽略，以便减少原型开发的成本。然而，缺少严格实践会使设计的软件原型降低成一次性模型。从根本上来说，不必在原型的开发上进行软件编码，应该在最终交付的软件产品上进行。

采用软件原型策略之前，应该先检查传统工程学科和软件社区对于原型的不同处理方式。表 1-3 提供了一个关于原型设计在传统工程和当下流行的软件实践中使用方式的对比。传统工程学科利用原型来支持实验和数据收集，以证明产品设计的某些关键特征。这里的原型并不是可以用于准备生产的模型，但是可以用于设计挑战和探索重要的设计理念。 25

表 1-3　原型策略的比较

传统工程	软件实践	缺点
概念验证原型用来测试预期设计中不打算进行精确模拟视觉外观、材料选择或制造流程的一些方面	快速原型是指创建的原型最后会被全部放弃，而不是成为最终可交付产品的一部分	任务原型需要不断调整、改进并且纳入最终产品
形式研究原型允许设计师们在不进行模拟产品实际功能或准确视觉外观的情况下，探究一个产品的基本尺寸、外观和感觉	不断演化的原型设计以一种结构化方式构建非常健壮的产品原型，并且不断地进行精炼	不断演化的原型设计认为某些需求难以理解，只需要关注那些容易理解的即可
视觉原型将捕捉预期的设计美学以及模拟的预期产品的外观、颜色和表面纹理，但不会真正体现出最终成品的功能	构建最终成品作为独立的原型，并将这些独立的原型在一个总体设计中合并	整体的软件架构是不存在的，并且直到原型被集成和测试之后，才能确定产品性能
全面的原型和最终的概念测试是工程师对设计缺陷的最终检查，并且允许在大规模有序生产前做出最终改进		关注于一个局部的设计原型会将开发人员从分析整个项目中分散开，致使他们忽视更好的解决方案，准备的只是不完整的规范说明，或者只能将局部的设计原型转化成低端产品，并且难以维护

⊖ 参看 http://en.wikipedia.org/wiki/Prototype.

（续）

传统工程	软件实践	缺点*
用户体验模型需要积极地人为互动，并且它主要用于评估潜在的用户如何与设计概念的多种元素、活动和行为进行交互		设计原型应该在短期内完成。开发者可能会尝试开发一个复杂的原型。用户的反馈可能会动摇原型中的某些细节，这会拖延开发团队并阻碍开发进度

* 参见 http://en.wikipedia.org/wiki/Software_prototyping.

软件行业正面临着原型设计的挑战，因为产品原型不缺少成品中所包含的任何物理特性。一旦软件代码编写完成并且演示功能正常，那就很难放弃这个原型并付出额外的努力进行高质量软件产品的设计、开发和测试。软件原型的构造在软件设计和编码标准方面不那么严格。这使得它们不足以被最终的可交付产品所采纳，并且将会增加与软件支持和增量开发相关的工作。软件工程师不倾向于从原型中总结教训，并且花费额外的时间通过设计、编码和测试重构软件组件。为了与设计和编码标准相一致，任务原型可能被“清理干净”，但是它的架构可能会禁止在未来进行修改或添加功能。最终，软件产品可能要经历一个完整的重构过程来检查设计局限性，以及来自因原型开发工作导致的不稳定结构框架的阻碍。

这部分的其余章节将会解决关于实际软件工程中软件开发环境的相关问题。这些问题包括通用开发项目框架、软件架构、如何应对项目复杂性以及集成产品和过程开发方法。第 6 章探讨了材料所面临的挑战，材料的应用使得软件产品工程相比于机械、电气、航空、自动化、热力学或化学系统来说更加复杂。此外，第 6 章还提出了软件工程原则和实践的概述。

26 ~ 28

通用软件开发框架

本书自始至终都在利用一种通用的软件开发架构来识别和考虑一个吸引人的软件工程命题。如果没有这种架构，就很难理解或表达软件工程，因为它在一个承担软件开发目标的项目环境中运行。这表明软件工程实践总是在一个软件开发项目环境中进行。讨论软件工程时不提出软件开发或项目管理的主题是不可能的。这意味着软件工程与软件项目管理是密切相关的。然而，因为它代表了一门技术学科，软件工程必须强调用于设计、实施以及维护软件产品的技术。因此，软件工程学科必须强调这种技术负担的本质，同时强调它与软件项目管理或大型系统开发项目管理框架的从属关系。从根本上讲，软件工程代表了建立技术工作和项目管理领域之间关系的监督功能。

本章给出了一套术语，用来表达构成软件工程知识分支的原则和实践内容。揭示并讨论了几个主要概念来启动软件工程调查。这些概念阐述了关于什么是软件产品、软件在一个项目背景下是如何使用的，以及如何利用一些基础管理工具来组织、监控和保证一个软件开发项目的顺利完工的基本观察结果。理解这些概念是很重要的，因为软件工程理论和原则依赖于这样的基础，并且将利用这些术语来表达。即使术语软件类似于术语产品，有明确的含义，但是前者却比后者更有影响力。因此，本书中使用的每个技术术语或表达式都有特定的含义，并且每个术语使用的指定含义都比另一个更精确。

一个通用软件开发框架的表达式是基于对术语软件的理解来推断的。类似于系统这个词，软件已经被赋予过多的含义，以至于失去了其作为交流工具的价值。因此，将要讨论术语软件，以使它在本书中的使用清晰明了。通过研究软件开发技术和实践，讨论软件开发框架来提出一个参考框架。当提到软件开发时，应该通过一些以项目里程碑和评审分隔的几个时间增量调用代表项目。软件开发共享项目和产品创意。软件产品的开发必须通过几个自然演化的阶段。由于软件产品的发展，软件开发过程涉及一系列开发阶段。软件开发的基础阶段是需求分析、设计、实现和测试。

开发过程 ⟶ 项目阶段 ⟶ 产品开发阶段

29
~
30

2.1 软件分解结构

要真正掌握一种软件产品的意义或本质，就一定可以理解它的结构。作为一种产品，软件不应该被视为为计算机处理器或数据处理系统提供指令的代码行序列。代码行代表了实现软件产品的媒介，就像用木材、砖、管道建造房屋一样。房屋在建造之前必须要进行架构和设计。软件是一种革命性形式的产品，因为从本质上来讲，它利用的是电子媒介。这使得软件是无形的，意味着它“摸不到”。[⊖]这并没有妨碍在部署或操作之前，软件产品需要进行设计、实现和测试。再一次用房屋作为参照，房屋设计的产生需要一个训练有素的“建筑师”

⊖ 参见 http://en.wikipedia.org/wiki/Computer_software.

来进行计划、设计并且监督建设。建筑师了解与建造房屋相关的工程挑战，并且会在房屋设计中使用某些合适的材料来适应该结构所能承担的结构外力和自然外力。根据完成的工程图纸，架构师准备了一份“材料清单”，上面列满了原材料和建筑产品以及用它们建造房屋所需的数量。

大多数的人造产品是由多种配件、组件或部分组成的。房屋的材料清单提供了建筑房屋所需的供应物和建筑产品的一个分项列表。因此，将这个概念应用到软件产品上也是合理的。然而，由于软件产品是无形的，因此它与特定的计算环境或用来执行数据处理的系统有着密切的关系。因此，如图 2-1 所示，通过计算环境来区分一个软件产品是不可能的。随着软件分解的不断扩展，它将为讨论软件开发框架提供一个基础。

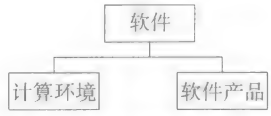


图 2-1　软件分解的第一层

软件工程关注的是软件产品的进一步组成。有许多术语用来识别组成软件产品的构建模块或元素，包括函数、程序、例程、子例程、应用程序和对象。这些术语来源于发明计算机语言的个人或团队，以及将代码翻译成机器可执行指令的编译器。使用不带有多余含义的术语来讨论软件分解是必要的。术语必须指明产品层次或结构配置的一个相对位置。这将借助组件和单元来完成。组件指的是一个部分、成分或者一个更大部分的组成部分。更重要的是，在每一版的百科全书词典中，组件都常被定义为一种组装，即“放在一起组成最终成品之前的一套组件”。从这些就可以推断出，软件组件代表软件分解结构（SBS）中组成最终产品的一个元素。组件也足够灵活，适合用来表示组装层次，如图 2-2 所示。

除了无终止的组件层次结构以外，这种结构概念都适用。因此，建立一个软件构建块是必要的，它可以终止层次结构。为此，将使用术语单元。软件单元代表了软件产品中最初被组装的软件组件中的基础元素或部分。百科全书词典中将“单元”定义为“尤其是为了分析，某些分离得到的个体、独立部分或元素”。因此，软件单元代表可以从中组装软件组件的一个软件分解结构中的独立元素。图 2-3 描述了一个软件分解结构的例子。

本书中提到的术语组件和单元指的是和软件元素层次相关的软件元素的位置。软件单元是一个基本的构建块，但不涉及任何分解或进一步分解。一个软件组件表示由两个或多个软件单元和 / 或组件组成的一个元素，如图 2-3 所示。

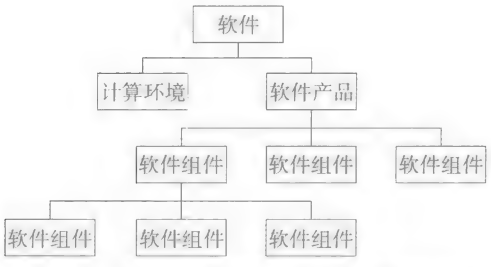


图 2-2　软件分解结构中的软件组件分层

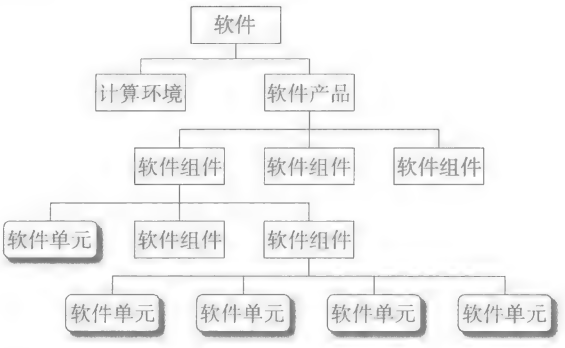


图 2-3　软件分解结构中的软件组件和单元的分层

分解视图中体现出的挑战是为什么有必要区分术语软件和软件产品。与计算环境关联对于传递与产品、组件和单元等术语相关的含义足够有效吗？通过讨论软件开发背后的概念能

获得更好的解释。就像建筑师解释房屋的设计一样，建筑工作中的其他方面都超出了房屋设计本身。必须考虑的事情包括在一块土地上定位房子的结构、与街道连通、水电安装以及污水排放。这些外围因素影响着房屋的设计及其结构，并且对于架构的总体性能和准确性来说必须是完整的。

这些外围因素必须在软件开发过程中得到解决，包括在产品发布、交付、安装和部署之后如何维护软件产品。知识的系统工程分支已经给出了一个完整的产品和过程开发概念来解释产品的并行工程和生命周期维护的概念。当这种模式应用于软件时，它将获取与开发后的过程相关的外围因素。就像软件开发可以称为开发过程一样，软件产品发布、用户操作培训以及产品支持和维护都可以称为过程。软件分解结构的集成产品和过程开发的概念的结合产生了图 2-4 中的软件层次结构。

31 ~ 33

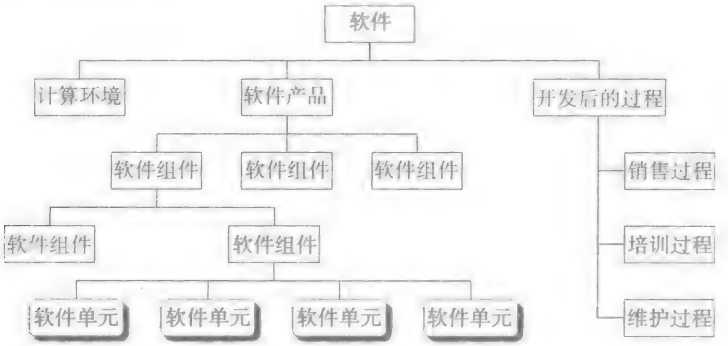


图 2-4　软件分解结构总图

开发后的每一个过程都能体现出在开发工作结束后的软件产品部署中定义、设计、实现，以及建立必要设施、计算环境、网络等需要付出的重大努力。这些过程必须及时进行，以确保当软件产品准备好部署时，它们是可用的。在软件开发工作中建立过程的这些工作应该作为次级项目，因为部分成员需要参与到软件工程集成产品团队中去，为软件产品的定义贡献他们的看法。这将需要包含软件开发的计划、预算和进度安排等工作。

2.2　软件开发过程

一旦出现在实验室的先进研究环境中，软件开发就会一直遵循着一系列阶段。最基本的一套阶段包括需求分析、设计、编码和测试。软件开发的这种简单表示方式与大多数的工程管理方法都相匹配，并且已经随着时间改进，因为与软件产品相关的规模、复杂性以及成本不断增加。将软件开发与这些项目管理准则和集成产品和过程开发准则相匹配生成一个框架，伴随着一系列的阶段、里程碑和评审，意在使软件适应成为独立的产品或者成为嵌入系统中的产品。图 2-5 描述了一个软件开发项目的概念性框架。它是概念性的，由于通过项目评审前的必要准备，即软件技术评审的识别，它背离了已经确立的文献。技术和项目级的评审是开发工作中的战略点，对于重要涉众的产品定义演化是有建设性的。里程碑代表一款新型软件产品开发中的重大成就，并且允许投资者来确定进行下一个开发阶段的价值。这些软件开发阶段、评审和审核都会在接下来的几节中进行描述。

34

图 2-5　概念性软件开发框架

2.2.1　需求定义阶段

在这个发展阶段，项目团队与涉众交互来收集、分析和优化关于待开发软件产品的一系列需求和期望。需求说明书是在产品交付时，准备用来介绍已商定的产品功能、特性和特征的文件。在开发后的软件维护过程（PDSS）中，最初的一组需求可以在产品说明或者一个或多个相关说明中获取。

1. 产品需求评审

产品需求评审（PRR）是一项对软件产品需求说明与参与软件技术和管理人员的技术性评审。其目的在于确保软件产品需求足够明确，并且利用确定的软件开发资源来共同完成和实现。产品需求评审是必要的，以确保软件需求说明和技术计划和进度表能够合理安排软件开发工作的成功执行。进行产品需求评审来为项目级的软件开发需求评审做准备。

2. 软件需求评审

软件需求评审（SRR）是一项项目级评审，通过涉众和项目管理代表来评审软件需求说明，并且收集关于需求、产品资格要求（测试、分析、检验和演示）、开发后的软件维护概念以及下一阶段开发计划的反馈。当该评审中所有执行项目或产生的评论都已圆满解决时，软件需求评审就完成了。

35

2.2.2　概要架构定义阶段

在这一开发阶段中，软件工程集成产品团队通过进行功能分析与分配，为软件产品建立了一个功能架构。可选的功能设计应该利用软件分析方法中用于执行风险评估和权衡分析的方法来评估和比较。模型、设计文档和初始数据字典都应该准备好。应该定义开发后的软件维护过程，并且需求应该记录在一个或多个过程规约中。

1. 概要架构评审

概要架构评审（PAR）是一项演化的软件架构的技术性评审，目的是确保架构方案已经为概要设计评审做好准备。该评审的重点在于软件产品的功能架构和初始结构化配置。这一技术评审涉及软件开发团队和关键涉众，目的在于评估概要架构定义，以确保它充分满足软件需求和涉众需要，并且是简单的（不复杂的），可以用已确定的软件开发资源实现。

2. 部署策略评审

部署策略评审是一项关于软件部署方法的技术性评审，该方法包括产品复制、包装、发

布以及必要时的安装和设置。这一策略评审确定了对于软件复制和发布的首选方法，以及这些开发后的过程的商业过程概念。该评审应该解决过程和支持人员在其业务中所必要的基础设施项目。每个商业过程中最初的需求说明都应该进行评审，并且在评审完成后基线化。

3. 培训策略评审

培训策略评审是一项关于软件培训方法的技术性评审，该方法用于培训和教授终端用户操纵软件产品。这一策略评审确定了对于软件培训和教授的首选方法，以及这种开发后的过程的商业过程概念。该评审应该解决必要的基础设施项目，以确保过程和需要的支持人员进行操作。软件产品培训过程中最初的需求说明应该进行评审，并且在评审完成后被基准化。 36

4. 维护策略评审

维护策略评审是一项关于软件维护方法的技术性评审，该方法用于为软件产品提供客户和软件支持。这一策略评审确定了客户和软件支持的首选方法，以及这些开发后的过程的商业过程概念。该评审应该解决必要的基础设施项目，以确保过程和需要的支持人员进行操作。软件客户和软件支持过程最初的需求说明都应该进行评审，并且在评审完成后基准化。

5. 概要设计评审

概要设计评审（PDR）是一项项目级评审，通过涉众和项目管理代表来评审软件功能架构，并且收集关于功能定义、性能配置、行为、数据定义、功能性说明以及下一阶段开发计划的反馈。开发后的软件维护过程的说明可能会进行产品概要设计评审或者由一组更有限的涉众代表进行独立评审。当该评审中所有可接受的执行项目或产生的评论都已圆满解决时，概要设计评审就完成了。

2.2.3　关键架构定义阶段

在这个开发阶段，软件工程集成产品团队通过进行软件设计综合，为软件产品建立了物理架构。可选的结构化设计应该利用软件分析方法中用于执行风险评估和权衡分析的方法来评估和比较。开发后的软件维护过程应该在这一阶段进行设计和记录。应该生成模型来验证每个过程处理预期需求的能力。

1. 详细架构评审

详细架构评审（DAR）是一项关于完整软件架构的技术性评审，目的是确保架构解决方案已经为概要设计评审做好准备。该评审的重点在于物理架构，确立了软件产品的结构化配置。这一技术评审涉及软件开发团队和关键涉众，目的在于评估详细的架构定义，以确保它充分满足软件需求和涉众需要，并且是简单的（不复杂的），可以用已确定的软件开发资源实现。

2. 部署设计评审

部署设计评审是一项关于软件部署过程设计和实现方案的技术性评审。该评审应该指定 37
所需的基础设施条目和人员来支持该过程。该过程的说明和设计文档应该作为基准，并且经授权的部署过程组织一旦最终批准，就执行实施方案。

3. 培训设计评审

培训设计评审是一项关于软件培训过程设计和实现方案的技术性评审。该评审应该指定所需的基础设施条目和人员来支持该过程。该过程的说明和设计文档应该作为基准，并且经授权的培训过程组织一旦最终批准，就执行实施方案。

4. 维护设计评审

维护设计评审是一项关于软件维护过程设计和实现方案的技术性评审。该评审应该指定所需的基础设施条目和人员来支持客户和软件支持过程。该过程的说明和设计文档应该作为基准，并且经授权的维护过程组织一旦最终批准，就执行实施方案。

5. 关键设计评审

关键设计评审（CDR）是一项项目级评审，通过涉众和项目管理代表来评审软件的物理架构，并且收集关于功能定义、性能配置、行为、数据定义、功能性说明以及下一阶段开发计划的反馈。开发后的软件维护过程的设计可能会进行产品关键设计评审或者由一组更有限的涉众代表进行独立评审。当该评审中所有可接受的执行项目或产生的评论都已圆满解决时，关键设计评审就完成了。

2.2.4　软件单元编码和测试阶段

在这个开发阶段，软件实现团队准备进行单元设计并且和集成产品团队执行同行评审。根据接受的同行评审，接下来单元设计会针对结构单元说明进行编码和测试。测试单元应该由实现 IPT 进行评审，并且一经接受，完成的单元就应该被组件集成和测试所获取。

1. 单元设计评审（同行评估）

每个实现单元应该进行单元设计评审来评估它的设计，以确保符合结构单元规约。该评
38 审应该由软件实现团队的高级成员和出席的软件工程集成产品团队中的代表一同来执行。

2. 单元资格评审（同行评估）

每个实现单元应该进行单元资格评审来评估它在测试条件下的操作行为以及它的结构单元规范的满意度。该评审应该由软件实现团队的高级成员和出席的软件工程集成产品团队中的代表一同来执行。

2.2.5　软件组件的集成和测试阶段

在这个开发阶段，软件实现团队执行组件集成策略。然后会针对结构化组件规范对集成组件进行测试，并且和软件产品团队一同根据集成测试结果进行同行评审。一旦被同行评审接受，就可以根据组件集成策略的需要，为进一步的集成工作提供集成组件，直到软件元素完全被组装起来并且集成在一个或多个软件配置项目中。

1. 集成准备评审（同行评估）

每个实现组件应该接受一个集成准备评审来达到以下目的：1）确保每个集成过程中涉及的实现单元或组件都已经圆满地通过了资格评审，2）评估集成方法，3）评估组件集成测试程序。该评审应该由软件实现团队的高级成员和出席的软件工程集成产品团队中的代表一同来执行。

2. 产品测试准备评审

在产品测试准备评审中，软件实现团队应该展示软件产品实现的状态，来提供它们准备软件实现中产品测试阶段的证据。软件工程集成产品团队的代表应该确保该软件产品集成测试已经完成，并且该软件测试程序以及环境都处于准备状态来支持产品测试。

2.2.6　产品测试阶段

39 在这个发展阶段，完成的产品应该通过验收测试程序进行测试。用于产品测试的测试环

境应该与验收测试环境一致，以确保在验收测试中不会出现未知的缺陷或失败。一旦产品测试成功完成，产品配置项目和它的相关文档应该为功能配置和物理配置审核做好准备。

1. 验收测试准备评审

验收测试准备评审是一项技术性评审，通过技术涉众和软件开发管理代表来评审软件产品测试的结果和来源于识别软件缺陷问题报告的状态。软件工程集成产品团队根据软件产品的准备来推荐软件开发管理人员，进入验收测试开发阶段。

2. 测试准备评审

测试准备评审是一项项目级评审，通过涉众和项目管理代表来评审软件预测试的结果和来源于识别缺陷的软件问题报告的状态。软件工程集成产品团队根据软件产品的准备来推荐项目管理人员，进入验收测试阶段。

2.2.7 验收测试阶段

在这个开发阶段，完成的产品要根据测试方案和程序进行测试。在软件不满足指定需求的情况下，软件工程集成产品团队就必须进行缺陷校正，或提出偏差申请或者放弃该产品。偏差表示对于一个软件产品的已知缺陷的暂时接受。不足之处在于要在软件产品即将发布时进行校正或者打补丁。偏差的目的在于允许对带有已知需求缺陷的软件产品进行发布或部署。放弃该产品是指涉众赞同允许不满足指定需求的软件产品不需要在即将发布或打补丁时进行软件缺陷的校正。

1. 功能配置审核

功能配置审核（FCA）是一项通过测试、检查、演示或结果分析核实软件产品是否满足功能基准文档指定需求的配置管理审核。该检查证实了所有授权的变更提议都被纳入产品和文档中进行优先验收测试。

2. 物理配置审核

物理配置审核（PCA）是一项针对已完成的（已实现的）软件产品技术文档对软件产品配置进行的配置管理审查。物理配置审核包括对于工程图纸、设计文档和规约说明的详细审查，以确保文档已经准备好支持开发后的过程。

3. 部署资格评审

部署资格评审是一项关于部署过程的技术级评审，以确保过程和人员准备好进行软件产品培训。该评审的结果应该在进行部署准备评审（DRR）时提出。 40

4. 培训资格评审

培训资格评审是一项关于培训过程的技术级评审，以确保过程和人员做好软件产品部署的准备。该评审的结果应该在进行部署准备评审时提出。

5. 维护资格评审

维护资格评审是一项关于维护过程的技术级评审，以确保过程和人员做好对客户和软件产品支持的准备。该评审的结果应该在进行部署准备评审时提出。

6. 部署准备评审

部署准备评审是一项用于评估开发后的过程及其文档的状态，并且准备向客户、消费者和其他涉众发布软件产品的项目评审。通过评审验收测试的结果来确保产品满足它指定的需求。通过评审软件产品审核的结果来确保产品以及相关文档做好支持开发后的过程的准备。一旦部署准备评审成功完成，软件产品就会从软件开发过渡到开发后的过程的操作强化状态。

2.3　总结

软件开发框架为软件开发和项目管理提供了一个结构化方法。涉众与软件工程集成产品团队的代表实现了一同参与整个开发工作的想法。集成产品和过程开发理论解释了开发后的软件维护过程的并行开发。必须及时建立这些开发后的软件维护过程来支持软件产品的部署和操作。软件分解结构包含了集成产品和过程开发理论，并且对开发后的过程中的项目计划、预算和进度安排都建立问责制。软件开发框架涉及一系列项目评审和审核，为项目管理团队和涉众提供一个机会来形成维护开发工作状态的意识。项目评审确定了进行每个后续开发阶段的技术准备工作。执行配置审核以确保已实现及测试的软件产品配置符合功能和物理基准。

在本书的支持下，软件开发框架为讨论软件工程实践的需求提供了基础。由于软件产品在整个开发过程中呈现出不同的表现，所以软件工程原则、实践和工具都必须谨慎使用。因为这一点，所以更多考虑如何明确地阐述软件工程任务，以解释软件产品和项目在开发框架中的状态。

41
~
42

第3章

Software Engineering: Architecture-Driven Software Development

软件架构

软件架构作为集成软件产品的表示，并为软件实现提供基础，本章对此进行了检查。由于软件产品的复杂性质，这里有几个描述软件产品所必须理解的观点。首先，一个软件产品的开发要满足客户和涉众的需求和期望。因此，软件产品需求需要被所有涉众认可，并且作为需求基线被纳入配置管理系统。这个基线允许针对基准化需求进行变更提议的追踪。

软件架构的第二个元素是描述了软件产品的操作过程、功能分解、性能和专业工程特性的功能架构。功能分解必须解决软件产品行为包括的几个方面，例如数据安全、错误和故障恢复以及相关安全操作。这些行为被错误地认为是非功能性需求。这些需求应该考虑由专业工程授权，必须用功能架构解决。故障检测和响应/回复功能直接影响了软件产品的可靠系数。产品复杂性、架构完整性和可恢复性直接影响了软件产品的可维护系数以及软件产品在未来版本中提高、巩固和功能扩展的能力。然而，这些非功能性需求是特定的，它们必须被纳入功能架构中去，并且在刺激反应的情况下，它们的行为在功能架构的总体表示中进行了明确的设计。

软件产品架构中最后一个元素是物理架构，它描述了软件产品的结构方面并且为如何将产品组装和集成成为一个或多个软件配置项提供了见解。物理架构在某种方式上来源于功能架构，涉及一个自上而下的概念模型和一个自底向上的表现形式。作为最初制定的功能架构，最上层物理架构中的结构组件可能是确定的。物理架构的基础来源于用于分组并合成来确定结构单元的功能单元。这暴露了结构配置中的一个解决建立软件集成策略的缺口。这种配置物理架构的方法将会在第12章进行详细解释。

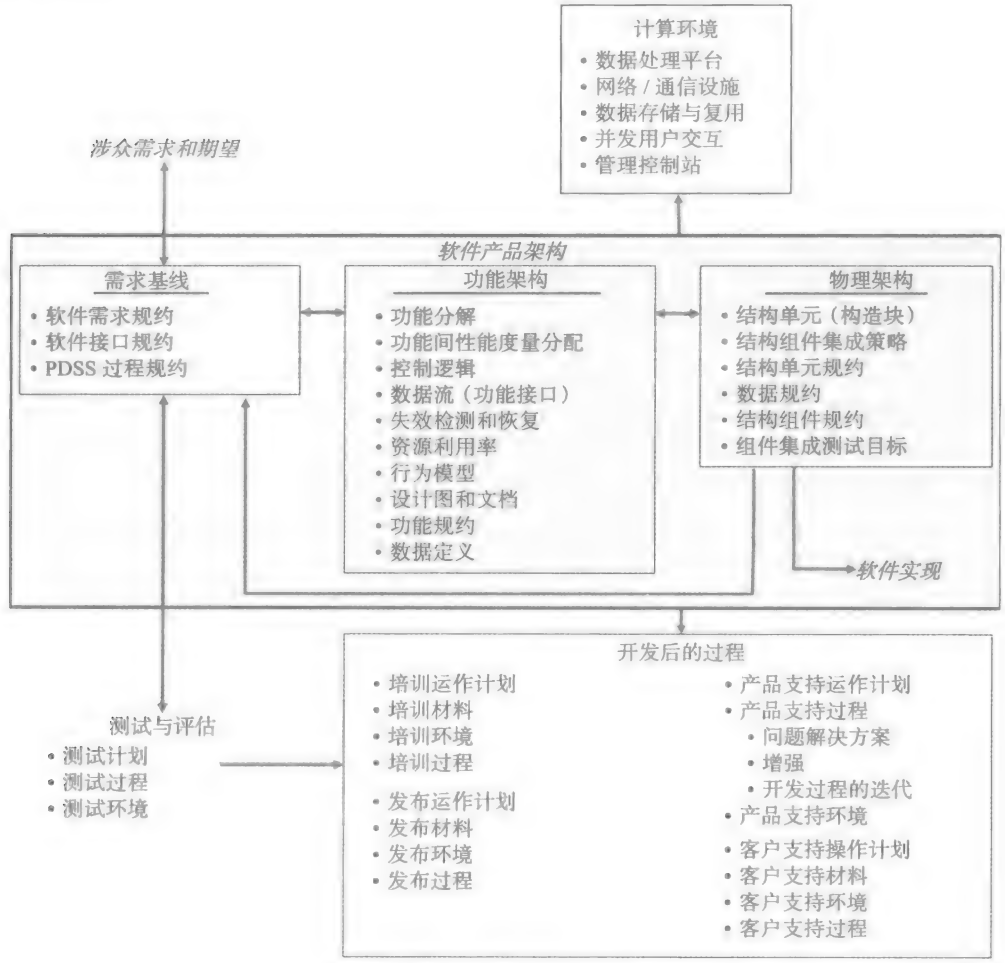
没有物理架构，软件实现工作就不能被正确地定义、计划和控制。软件工程集成产品团队负责开发和控制软件架构以及它的集成设计和配置文档。软件架构必须表现待开发的软件产品的设计。这需要绘制描述软件架构的不同类型的设计图表、视图和文档。设计文档的一般分类包括：

- 功能架构的描述和它的设计表示，例如功能规约、功能分解层次结构、数据流图、行为模型和数据字典。
- 物理架构的描述和它的设计表示，例如界面分块图、结构规约，以及配置组装和集成方案。
- 需求基线和它的表示，例如软件需求规约、软件接口规约和数据库需求规约。
- 计算环境的描述。
- 开发后的过程的规约和设计文档。

在这些设计描述之中存在一些关系和依赖性，必须进行协调和同步来使项目能够考虑潜在的软件设计机遇，并且通过合并软件架构中已许可的变更来应对变更请求和建议。

软件架构、计算环境中的元素以及存在于这些元素中的关系和依赖性都能在图3-1中识别出来。这些关系是通过软件架构中任意两个元素之间的箭头来表示的。这些关系代表了源

元素与目标元素之间的依赖关系。这些关系和它们的依赖性将在下面几节中进一步检查。注意到涉众需求、软件实现以及测试和评估元素在图中显示为灰色，表明它们不是软件架构中
44 的元素。它们都包含在图中，因为它们是建立软件架构所感兴趣的，并且它们也是软件开发工作涉及的。



45 图 3-1　软件架构元素

定义软件架构是软件工程集成产品团队的责任。这个多学科团队包括来自软件开发工作中涉及的所有技术组织的代表。这些代表们带来了他们的技术知识和组织兴趣在软件架构的定义上进行合作。软件工程集成产品团队负责制作和维护构成产品数据包的图表、图纸、模型和文档。产品数据包提供必要的信息，使得软件实现团队能够去设计、编码和测试（制造）结构单元，并且在一个或多个完整的软件配置项中组装、集成和测试结构组件。软件数据包可以通过已实现的附加文档来增强，这些文档在软件的实现过程中产生，是软件产品维护所必需的。软件产品配置审核执行在部署准备评审之前的数据包，以确保产品数据包能准确地反映产品实现并且合并所有授权的变更提议。

由于软件架构中元素的这些关系和依赖性，需求管理的实践和可追溯性必须扩大来广泛集中解释其必要性以支持软件开发和产品维护。软件需求涉及软件产品、外部系统接口、计

算环境和几个开发后的过程。从涉众需求到软件产品需求、功能和物理表示以及开发后的过程中存在的一系列可追溯性，对于解决方案的实现是至关重要的。图 3-2 描述了关于软件产品和开发后的过程的需求追溯链。这种关系和依赖性将在接下来的章节中进行确定和讨论，以强调架构元素之间的关系以及如何在软件产品配置中将它们合并。

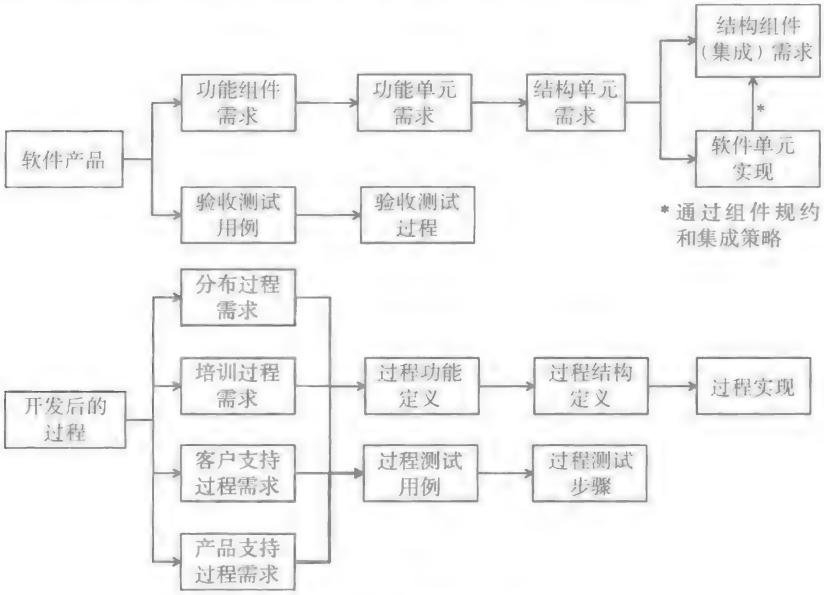


图 3-2 需求追溯链

3.1 涉众需求的关系和依赖性

涉众在软件产品的开发、操作和维护中有既得利益。他们从其职业、个人事务或娱乐中 46
存在的产品获得利益或效用。涉众需要建立价值、潜力、动机以及产品存在的结果并维护。涉众根据其职业、企业和行业对产品的组成、特点和应用贡献着他们专业的辨别力。当产品最终部署完成，涉众获得与产品开发密切相关的个人利益和满足感。此外，涉众往往成为倡导者，在他们的商业投机、产业或社会圈子中使用该产品，并且他们为开发出一个成功产品所做出的贡献深感自豪。涉众需求和软件需求之间的主要依赖关系为：

1）产品业务特征（需求基线）。涉众期望在他们的职业、个人兴趣或休闲娱乐兴趣中使用该产品。他们将提高软件产品的操作特性，使产品的好处、优势或娱乐价值可以被充分利用。

2）产品性能特征（需求基线）。涉众对效果、响应能力以及执行数据操作、计算和演示的效率感兴趣。

3）产品物理特征（需求基线）。涉众比较关心软件产品的风格、精致程度和美学品质，因为他们对此有权威。整个用户社区巧妙地回应了图形展示、用户界面以及数据表示的吸引力、创意和创新的“外观和感觉”对于验收的影响。

4）产品资格需求（需求基线）。涉众必须同意每个将被验证的需求的产品资格方法。典型的资格方法包括分析、演示、检查和测试。每种方法都以这种方式规定，对测试结果进行分析来判断产品是否符合指定的需求或者展示出期望的特性。

5）产品接口需求（需求基线）。涉众都受到用于支持它们商业操作和专业任务的业务系

统或其他产品的存在的限制。产品接口提供了系统或产品中数据的无缝交换，并且这些产品接口不能被任意修改来促进软件产品的设计和实现。

6）开发后的过程（需求基线）。开发后的过程的需求是根据软件产品的定义而确定的。用于打包和分发产品的方法将通过行业规范和竞争方法来确定。用户培训的类型将依赖于产品的复杂性、它支持的商业或业务任务以及自身接口连接外部系统或产品的数量。可用的客
47 户和产品支持的类型和数量取决于客户基础的量级和业务或个人对产品的依赖来实现他们的专业事务。

3.2　软件需求基线的关系和依赖性

软件需求基线由许多被协调在一起的规约组成，它们通过软件产品、计算环境以及相关开发后的过程来清晰地表达预期的功能、性能、物理特征和质量特征。这套需求关注的活动是复杂的软件架构以及验收测试的场景和程序的定义。在功能和物理配置审核过程中，该需求被用作标准，针对评估的软件产品实现来确定它的工艺质量。需求基线包括以下涉众需求或其他软件架构元素的关系和依赖性：

1）开发成本和时效性（涉众需求）。软件需求将最终确定开发工作的成本，并且影响产品交付的里程碑。确定如何平衡软件产品需求以便开发项目保持稳定，并且能够果断地向结果推进，取决于为项目成功确立条件的需求基线的准确性和合理性。

2）软件产品需求是否合适（功能架构）。产品需求的适用性、正确性、彻底性和精密程度将影响开发工作。需求的广度将影响该产品在市场上的成功。软件产品需求会被解释和转化在功能架构中。就有序的安排、复杂性和适应性而言，含糊的或者模棱两可的需求可能会掩藏功能架构中的最优组合。被夸大的需求将抬高开发成本并且可能会增加产品的复杂性。

3）测试和评估工作的范围（测试和评估）。需求基线同时确立了软件产品和开发后期软件维护过程的测试和评估工作的范围。允许过多的需求将大大地增加测试用例和场景的数量和复杂性。测试工作可能会在预算、工具、设备和进度安排方面消耗大量的项目资源。

4）开发后的过程的范围（开发后的过程）。增加的产品复杂性会影响用户的易用性和培
48 训需求，以及产品和客户的支持成本。

3.3　计算环境的关系和依赖性

计算环境涉及计算机械设备、数据存储设备、工作站、软件应用以及支持处理和交换软件解决方案所需的电子信息的网络。计算环境包括下列软件架构中元素的关系和依赖性：

1）技术可用性（需求基线）。软件解决方案的执行被计算环境所制约，并且必须被分解成软件产品需求。可以执行的指令数量、数据的传输速率、图形分辨率和显示率是典型的计算设备措施，影响软件解决方案的后续性能。

2）资源的利用和保护（软件产品架构）。计算环境中计算资源的可用性将限制软件产品的性能。必须开发共享的资源利用模型，尤其是网络多用户应用程序。必须开发一项策略来管理资源，即建立资源消耗、保护、保存和恢复的策略，并且在软件架构中进行合并。

3.4　测试和评估的关系及依赖性

测试工作可能会在预算、工具、设备和进度安排方面消耗大量的项目资源。测试和评估工作决定了软件产品的适用性和与需求基线相关的开发后的过程。测试和评估工作包括下列

软件架构中的关系和依赖性：

1）测试覆盖率（软件产品架构）。软件测试覆盖率为软件实现的进展提供了一个重要指标。针对软件架构中这三个元素，追踪测试覆盖率以确保软件产品在测试中完全暴露是很必要的。在软件实现的过程中，重点在于跟踪软件单元和组件测试覆盖率体现出的有多少应用程序是你正在执行的。在测试计划中，根据测试的需求关联测试用例和场景是必须需的。

2）测试充分性（需求基线）。测试软件产品需要一些深思熟虑的审查来确定一套能识别的测试用例和场景，它们将在简单、实用并且极端的条件或情况下测试软件产品。边界测试 49
关注软件需求的边界条件。应该执行边界值分析以确保这套测试用例足够证明软件产品需求会接受足够的测试。

3）开发后期软件维护过程的测试工作（开发后的过程）。每个开发后的过程都需要测试和评估工作，以确保一经发布，过程就已准备好支持软件产品以及其客户或用户。开发后的过程与软件产品架构密切相关。

4）回归测试（需求基线）。在软件实现过程中，已经经过测试的现有编码需要在修改完成后重复测试，例如功能增强、打补丁或更改配置。正式的变更提议和技术变更请求必定在回归测试中占有一定数量，通过变更提议解决结构级的组件集成需求是必要的。

3.5 功能架构的关系和依赖性

功能架构描述了软件产品为了满足指定需求而必须执行的事务。为了确定这一点，必须分解顶级功能来识别子功能序列、控制逻辑和决策分支，以及使用每个函数必要的输入和输出。功能可能需要计算资源，例如数据存储或数据传输带宽，才能有效地执行。功能架构包括以下软件架构中的关系和依赖性：

1）功能行为验证（需求基线）。在功能行为定义中，软件需求的功能分解结果表现了软件对可辨识刺激的回应。这些行为描述了实时的功能流、数据流、控制流和每个用户交互的资源利用率、软件操作和接口事务。这些行为模型的准确性和适用性必须确认为是软件需求的一个合理解释。

2）性能分配确认（需求基线）。功能性能的预期范围必须被证明能满足软件需求指定的性能等级。基于计算环境供应商发布的设备特征，功能性能预算应该是合理的。

3）功能规约的完整性（需求基线）。完整的功能架构必须被证明能解决所有的操作或系统状态和业务模式、用户输入、接口信息配置文件以及故障检测、隔离和恢复解决方案。 50

4）功能性同化（物理架构）。每个功能单元都必须被分配给一个能够与类似功能结合、集成和混合的结构单元。由此产生的结构单元需求必须能被每个功能单元规约追踪到。

3.6 物理架构的关系和依赖性

物理架构描述了软件产品的布置、接口、组装和集成的配置：

1）结构设计验证（功能架构）。必须验证物理架构来确认所有的功能需求都已纳入结构单元和组件集成规约中。

2）结构设计优化（功能架构）。必须证明物理架构设计的改进和优化符合功能规约。结构设计改进可能会包含适当具有谨慎和智慧的结构设计，并且可能需要进行功能规约调整。

3）结构性能验证（需求基线）。结构设计估计的性能必须确定使用静态数学或动态模型。必须证明这些性能估计能够满足指定的需求。

4）产品性能度量（软件实现）。必须针对预计的结构设计性能，对软件实现的实际性能进行测量和验证。

3.7　开发后的过程的关系和依赖性

开发后的过程涉及它们自身的特定开发过程，但是这并不是这本材料的主题。然而，由于与软件产品密切相关，它们的需求被指定为软件产品开发工作中不可或缺的元素。因此，这里有意将认定的关系一般并广泛化：

1）过程有效性验证（需求基线）。在程序和员工资格培训方面，每个开发后期软件维护过程（PDSS）都必须进行验证以确保操作的准备和有效。

2）过程完整性（软件架构）。必须对每个开发后的过程中的全部定义、设计和实现都进行验证来与软件架构保持一致。

51

3.8　软件架构的动机

软件开发有一段关于项目和产品失败的污点史，并且还获得了无纪律、幼稚的工艺这样的名声。虽然犯过很多错误，改进实践状态的大量尝试也都失败了，在软件开发效率、有效性或产品质量方面也没做出任何重大的提升。过去失败的尝试可以归因于未能设计一个解释产品需求、功能和性能特征，以及组装和集成结构注意事项的警戒设计范例。软件架构，就像在这里解释的一样，目的是避免软件设计的不利因素。

软件架构包括大量的与集成产品和过程开发理论相一致的元素或子架构。下面列表识别了在软件架构中找到的各种子架构元素。注意，对于每个最低级子架构而言，都有三个子元素：需求基线、功能架构以及物理架构。

软件架构
- 产品需求基线
- 产品功能架构
- 产品物理架构

软件开发后的过程架构

产品发布过程架构
- 发布过程需求基线
- 发布过程功能架构
- 发布过程物理架构

产品培训过程架构
- 培训过程需求基线
- 培训过程功能架构
- 培训过程物理架构

产品维护过程架构

用户支持过程架构
- 用户支持过程需求基线
- 用户支持过程功能架构
- 用户支持过程物理架构

问题解决过程架构
- 问题解决过程需求基线
- 问题解决过程功能架构
- 问题解决过程物理架构

产品改进过程架构
- 产品改进过程需求基线
- 产品改进过程功能架构
- 产品改进过程物理架构

该工作自始至终讨论的软件架构是基于系统工程实践和原则。软件架构包括其他典型工 52
程学科中的设计工件，例如图表、图纸以及动态和静态模型。典型软件架构工件如图 3-3 所示。该架构被用于控制产品配置和促进产品设计框架所包含的授权变更。它解释了软件产品复杂、非物质的本质，以及它与产品管理工具和技术的关系。与物理架构相关的工件形成了软件技术数据包，提供给软件实现团队进行软件实现的设计、编码、集成和测试。

图 3-3 典型的软件产品架构工件

每一个复杂的产品开发活动都涉及源于工程原则的某种形式的产品架构。将以下定义提供给术语**产品架构**：[⊖]

一个产品或系统的功能元素的描述的方法取决于它的构成组件或子系统，以及它们交互的方式。

这个定义标识的主要意图与软件产品架构相关——描述的方式取决于物理元素的功能以及它们如何交互。这意味着产品架构包括两个不同的设计表示，一个是功能上的，另一个是物理上的。产品功能和性能目标源于产品需求。功能分解将一个复杂的问题空间分解为更小的、降低难度的任务，可以通过更少的挑战来解决。功能分解在其最基本的形式中，通过在构成的功能中预计的性能特征来表示一个层次分解。功能架构描述包括函数的输入和所需的资源来产生函数的输出或提供的服务。最低级的函数、功能单元归属于在功能和物理架构领 53
域确立关系的物理设计中的结构单元。结构组件集成策略为一个或多个软件配置项中结构单

⊖ 参见 http://www.businessdictionary.com/definition/product-architecture.html.

元和组件的组装和集成建立了方法。

本章讨论的软件架构包括软件产品架构、计算环境和开发后的过程的定义。软件架构解决了产品以及与产品操作和维护密切相关的附加元素。这些构成了软件架构的每一个元素都涉及一套需求以及一个功能和物理表示。这与之前解释的软件分解结构和集成产品和过程开发理论是一致的。软件产品架构包括产品需求、产品功能架构和产品物理架构。软件产品架构中的元素将在本书的以下章节中进行更详细的考察：

- 产品需求——第 7 ～ 9 章
- 功能架构——第 10 章和第 11 章
- 物理架构——第 12 章和第 13 章

理解软件项目环境

有效且有利的执行软件工程项目涉及对于项目环境固有的复杂交互和依赖关系的理解。这种认识必须通过一套可以提供关于任务和工作产品现状信息的监管工具来加强。这信息包含了威胁到项目成功或软件产品质量以及市场竞争力的潜在状况的模糊征兆。软件工程利用这些信息使细心的实践者认识到破坏性的趋势，并且以一种积极的回应方式来消除有问题条件的根本原因。

有3种用于指导项目成功完成的基本管理工具。第一种是集成总体规划（IMP），识别组织角色和职责、要执行的任务以及预期结果。第二种是集成总体进度安排（IMS），提供关键事件的时间表、里程碑、评审和决策点。最后一种是项目预算，识别分配给每个组织来确保计划任务执行的资源。然而，必须正确开发、监控以及调整这些项目管理工具来反映任务评 55
价中固有的歧义。初步规划对于预期生产力、性能和结果的预测必须对项目的不确定性做出解释。对实现项目目标的自信程度包括对于生成项目规划和可能结果中涉及的假设的理解，如果该假设证明了有缺陷或不准确。当进展缓慢或受到不可预期状况的阻碍时，软件工程团队可以通过了解关于项目规划和预期恢复策略假设和决定的影响来控制项目的命运。

项目规划、时间安排和预算仅仅制约了用于限制项目团队及时交付产品和成本效益方式的设备。计划在最初是不准确的，因为关于待开发的软件产品有太多的未知数。此外，也不可能预测未来或意想不到的事。软件开发项目涉及的涉众的数量使得几乎不可能建立一个准确的项目规划。因此，使用软件工程实践最重要的动机是填补关于软件产品理解的空缺。反过来，这也有助于改进技术、项目规划、时间安排和预算，使得项目可以成功执行。

建立软件开发项目的目标是向一个或更多客户交付一个“新”软件产品。因此，项目规划将一直是不精确的，直到软件产品的定义相对完整。这意味着项目规划、时间安排和预算都仅仅是指导项目团队进行软件产品定义、设计、实现、测试、记录和交付的工具。项目团队所面临的困境是确定如何以这样一种可以满足项目目标的方式来定义软件产品。在这种情况下固有的事实，也就是项目规划、时间安排和预算，仅仅是一种达到按时成功交付软件产品的方法（根据时间安排），并且授权资金不超过阈值（根据预算）等目的的手段。只要项目团队可以在交付日期交付一个令人可接受的软件产品，并且消耗的资源不超过授权，就应该认为该项目是成功的。

在项目环境中存在各种决策点，表示有机会维护项目范围，以便能够实现目标。通过结构化软件工程实践和工具来识别在软件产品定义时提供的重新审查项目计划的机会。在每个机会中，都必须在可选方案中做决定。做出合适的架构设计决策涉及以下因素：

1）理解产品功能和特征对于涉众来说是很重要的（需求基线）。

2）确定如何提供每个产品特征（功能分析和设计合成）。 56

3）辨别哪种设计方法能够为当前涉众以及预期涉众团体或客户群服务到最好（权衡分析）。

4）消除未知条件可以提高实现项目和产品目标的可能性（风险评估）。

5）确保每个功能或特征对于产品操作是必要的，并且不超过需求（验证和确认）。

6）控制产品复杂性来简化软件业务并支持成本（集成产品和过程开发）。

7）改进技术和项目规划、时间安排和预算来反映选择的行为（控制）。

从根本上讲，软件产品架构决定了项目工作在产品生命周期中必定能成功地实现、测试、交付和支持。如果允许项目定义来驱动软件产品定义，那么该产品在一个竞争激烈的环境中可能就不太有利，也不太值得注意了。项目范围必须与向客户定义、设计、实现、测试和交付的软件产品提供的必要资源（人员、设施、设备、工具、预算、时间安排等）相一致。软件产品的开发必须适应包括用户、支持员工、培训员工、投资者和企业管理在内的所有涉众的需求和期望。当产品定义和项目范围不平衡时，那么软件工程、技术和项目管理团队必须相互协作来稳定局势。

软件工程工作代表了项目范围内全部的技术工作。因此，软件工程的领导阶层负责用一种与项目范围一致的方式来定义软件产品架构。当发现通过额外的项目资源，该产品对于其客户（消费者、经营者、投资者等）的价值提高了，那么就会生成变更提议来建立该增值的优势。在确定的项目成本和进度目标之内，无论何时都不能适应该增值，这种情况就会发生。图 4-1 描述了一种项目环境下的软件工程扮演的角色。

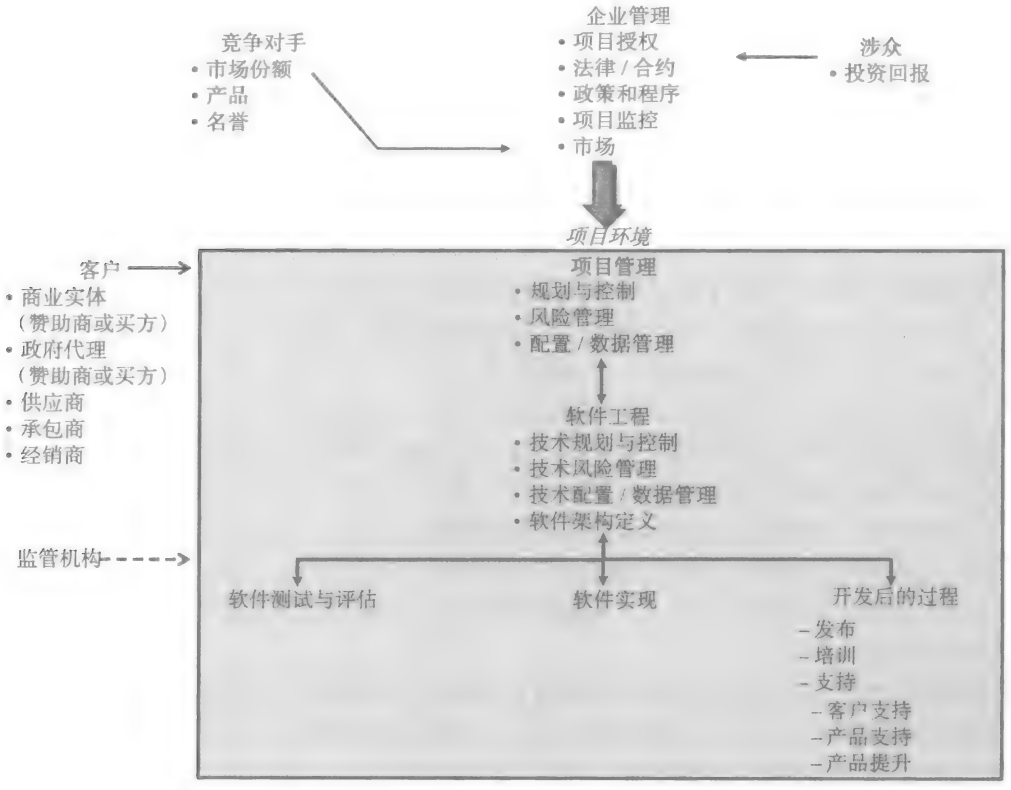


图 4-1　软件工程在项目环境中的角色

软件开发工作的复杂性可以通过分析大量的产品来识别，这些产品必须要通过开发项目才能解决。图 4-2 提供了一个列表，关于通过软件开发项目内部 6 个组织共同产生、协调和

控制而成的重要工作产品。表 4-1 确定了分配给每个软件开发项目组织的工作负载。软件工程组织直接负责全部工作计划的四分之一。然而，软件工程领导了全部的技术工作，包括其他技术组织的工作产品——除了项目管理产品之外的所有东西。这占了工作的 77.5%。此外，作为领先技术的代表，软件工程组织代表为项目管理工作产品的生成、协调和控制提供帮助。

图 4-2　软件开发的工作产品

表 4-1　软件开发项目组织工作负载

组织	工作产品数量	工作计划百分比	组织	工作产品数量	工作计划百分比
项目管理	20	22%	计算环境定义	9	10%
软件工程	21	23%	开发后期软件维护	22	24%
软件实现	10	11%	总计	91	
软件测试和评估	9	10%			

很少或没有软件工程实践的软件开发项目几乎都没有机会成功。进行软件开发却没有一

个有效的软件工程方法，类似于建房屋没有建筑材料和工程图纸。没有完成建筑图纸就开始建造房屋是不明智的。那么为什么没有可用的软件产品架构就开始编码就可以接受呢？软件工程实践为软件开发项目提供了以下优势：

- 在全部技术工作和项目目标之间保持平衡。
- 建立软件产品架构。
- 控制软件产品的复杂性。
- 与所有涉众协调变更提议和请求。
- 协调组织工作计划来维护校准项目目标技术计划。

本章的其余部分确定并讨论了有助于实现软件产品和项目校准的软件工程实践和工具。

4.1　集成产品团队

集成产品团队（IPTs）是用于管理软件开发工作复杂性并确保合适涉众参与决策过程的组织机构。集成产品团队是组织开发人员和涉众代表进行集成产品和过程开发的基础。集成产品团队的实现表示从一个烟囱式的功能组织安排向产品和过程关注点的转变。团队合作驱使功能学科进入一个相互支持的关系，这有助于消除阻止软件成功开发的障碍。团队可以在各级组织中形成，并且有权对软件产品的开发或维护过程做出重要的生命周期决策。图 4-3 给出了几个推荐的软件集成产品团队。根据本文的关键本质，应该组成额外的集成产品团队来监督个人产品元素的发展或维护过程。

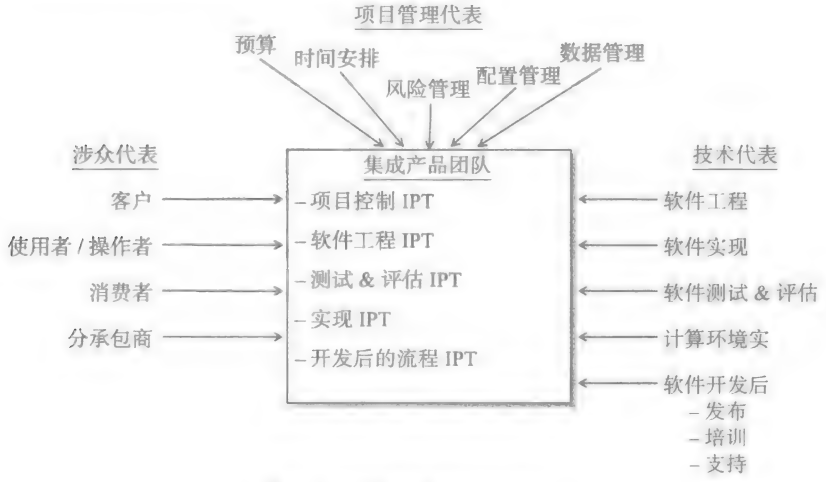


图 4-3　推荐的软件开发 IPTs

软件工程集成产品团队由首席软件工程师主持，并且包括来自技术组织和涉众群体的代表。团队负责以一种能确保所有代表的利益都均衡和谐的方式来建立和控制软件产品和过程架构。这个团队无法解决的问题应该提交给项目控制集成产品团队审议。它作为可纳入当前开发项目范围的变更请求的技术变更控制委员会（CCB）。不能在可获得的技术资源水平下完成的变更请求应该提交给项目控制集成产品团队考虑。

项目控制集成产品团队负责监督关于项目目标实现的项目进展，以及涉众对软件产品的满意度。作为变更提议的过滤器，团队需要额外的资源来执行应用程序。在可获得的项目资源水平之内，应该将不能适应的变更提议提交给项目的技术变更控制委员会。

4.2　软件架构

软件架构确定了与软件产品相关的业务和商业需求和特征。软件工程建立了一个软件架构来解决软件产品的功能和物理配置，并且影响着开发后的过程。因此，软件架构包括软件产品架构和每个开发后的过程的架构。图 4-4 展示了软件架构和它与软件产品和开发后的过程架构的关系。

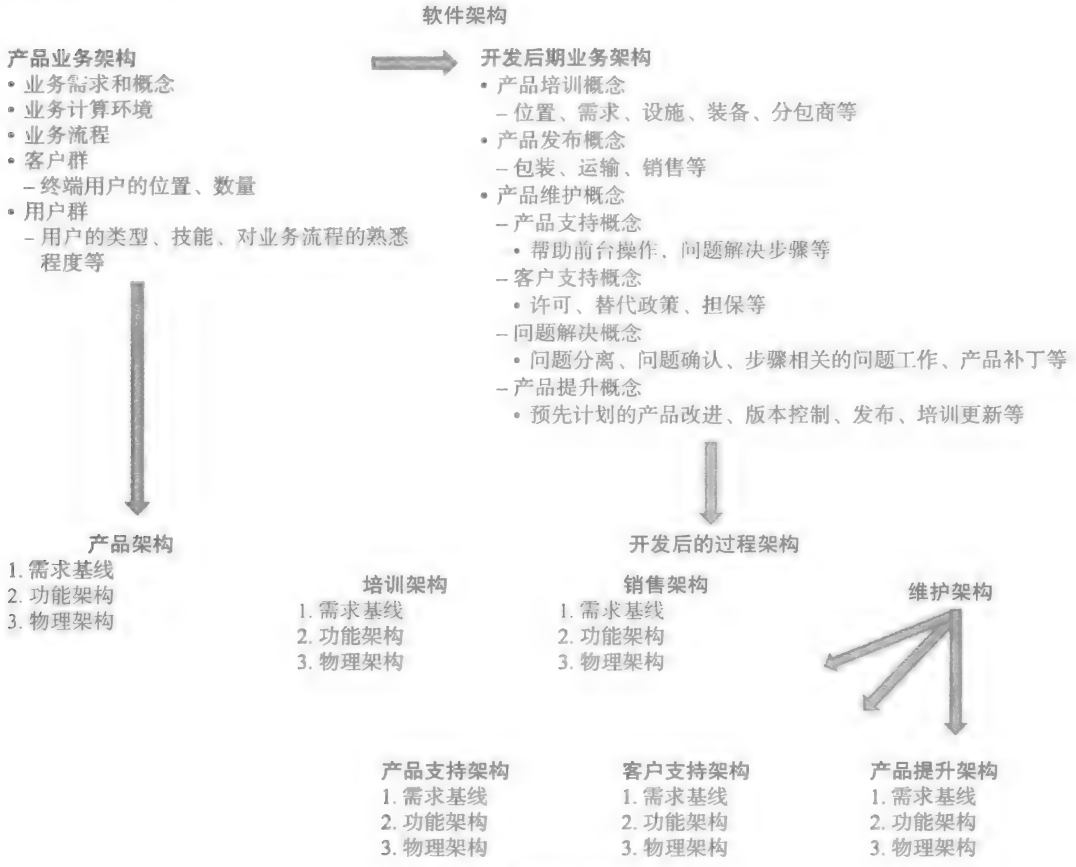


图 4-4　软件架构元素

每个架构都包括 3 个不同但集成的视角：需求基线、功能架构和物理架构。*需求基线*表示能够解决软件产品或开发后的过程领域的一套需求规约。*功能架构*表示功能组件和单元中对于功能和性能需求的分解。可以开发一个或更多行为模型来描述循环和业务规则方面的功能流、数据流、定时和执行逻辑（例如，If…Then…Else…，Select…Case 结构）。规范中的每个功能元素都是有特征的，可以提供关于它性能、行为和对于错误条件处理的细节。*物理架构*表示软件产品或开发后的过程领域的结构设计。这来源于最低级别的功能单元，并且解决了指导软件产品或开发后的过程领域的组装、集成和测试的组件集成策略。

1）*软件产品业务架构*。产品业务架构描述了关于考虑开发的新式软件产品的业务和市场要素。这就应该提供关于业务计算环境的信息、它支持的业务流程、行业的客户基础以及潜在企业数量和使用该产品的最终用户。应该建立最终用户的类型和相关技能水平的分类来提供建立软件产品培训和帮助系统的基础。必须解决产品的授权、担保和退还政策。

2）开发后期业务架构。开发后期业务架构应该确立预期的方法来提供软件产品的发布、安装、培训和支持功能。应该考虑现有设施的使用、次承包商以及与企业相对的软件代理商直接销售、培训和进行员工支持。

57 ~ 62

3）软件产品架构。软件产品架构确定了产品需求以及将被实现和测试的功能和物理特征。第3章提供了一个关于软件产品架构的详细描述。

4）开发后的过程架构。应该对每个开发后的过程进行定义来解释如何发布、维护软件产品以及提供最终用户培训。

- 培训过程架构。在培训过程、材料、设备以及为最终用户理解和精通软件产品操作提供帮助等方面，培训过程架构应该识别培训计划。第三方培训机构的使用、客户培训人员的培训或企业培训能力都应该被考虑。
- 发布过程架构。发布过程架构应该识别软件产品运输、安装和切换到业务状态的各种方法。软件媒体内容（可执行文件、安装方式和用户手册）都应该被识别，并且应该解决媒体的发布方法。应该对使用与电子媒体相对的盒装媒体交付进行评估，来确定给客户传送软件产品和手册的最佳途径。
- 维护过程架构。维护过程架构应该识别必须提供的产品和客户支持的类型。应该定义问题解决过程和程序，并且指定产品维护的设备、工具和人员需求。应该对开发和分布问题的补救方法和产品改进的预计划进行描述并标注其特征。

4.3　复杂性控制机制

软件工程工作主要关注于产品和项目复杂性之间的关系。使用一些软件工程实践来管理这种关系并监控实现项目目标的进展。将对这些复杂性控制机制中的每一个进行讨论，应用到软件产品中去。然而，它们也同样适用于计算环境和开发后的过程的定义。

4.3.1　工作分解结构

工作分解结构（WBS）将工作活动分解为可管理的、可以表示为工作包的任务。每个工作包都确定了必须完成的任务的序列，来实现一个中间工作产品或结果。工作包确定了来源于完成该任务所需的其他资源、劳动力、会议、差旅费用和物质资源的输入。工作安排确定了预期的开始日期、任务开始的必要条件、任务执行的时间间隔，以及预期的完成日期和确定当推迟完成任务开始影响其他任务的截止日期。

63

定义工作分解结构最常用的技术是使用产品分解结构来确定定义、设计、实现和测试产品各个部分的工作必要性，并进行组件的装配、集成和测试。这允许每个项目组织来识别与软件产品中的每个元素相关的工作，确定员工的工作，以及确保员工技能和专业知识能够正确地应用在所有工作产品中。应该为每个工作包分配一个工作授权或成本核算控制编号来随时追踪成本收益。

工作包的定义在提供必要信息来构造工作计划、时间安排和预算的任务执行之间建立了依赖关系。工作计划应该确定这些为产生工作产品、生成组织角色和为每个任务做出贡献，以及执行确定工作所必要的工作水平和资源所必须完成的活动和任务。应该确定任务执行风险，并且纳入工作计划和时间表中，以识别必须监控的潜在问题状况，并且预期应对不利事件。

工作分解结构一直是用于定义、组织和追踪进展以实现项目目标的项目管理工具。它通过捕获必要的待执行工作来交付软件产品和建立开发后的过程。它说明了软件开发组织中的

众多交互都需要在开发活动中协调和合作。然而，一款新的软件开发产品不能在开发的早期阶段以它的产品分解结构来定义。因此，必须不断地重新审视并阐述工作包和成本分配，以反映软件架构的演化。

工作分解结构提供了一个极好的项目度量和产品复杂程度的关系。任务之间的一些依赖关系为实现项目目标建立一条关键路径提供了基础。松弛时间（完成计划和截止日期的间隔时间）表明了如何压缩以构建好的工作计划。关键路径上元素之间的松弛时间越少，表明了关键路径上的相对于任务将按照计划完成的压力和意义。应该将松弛时间安排在时间表的关键路径上，来解决可能导致任务延期的意料之外的事件。

4.3.2　产品分解结构

产品分解结构（PBS）表示了软件产品分解成软件配置项、多级结构组件和结构单元。产品分解结构是至关重要的，因为它为软件组件的集成和测试提供了基础，并且细化了工作包和预算分配。然而，最初的产品分解结构只能解决最高层已知的配置项和软件组件。由于软件产品架构的发展，最初的产品分解结构变得越来越准确。

64

产品分解结构完整的定义是从软件产品物理架构中提取的，并且反映了设计、编码和测试软件单元以及将这些单元集成到软件组件和配置项中的工作。因此，支持早期的项目和技术规划的最初产品分解结构是基于表示大楔形预期工作的抽象软件组件。这些初步规划近似被称为大致预算，并且提供一个工作计划中的占位符，随着可以获取到更多关于软件产品架构的知识，工作计划的准确度也势必会提高。图 4-5 描述了产品分解结构随着软件产品物理

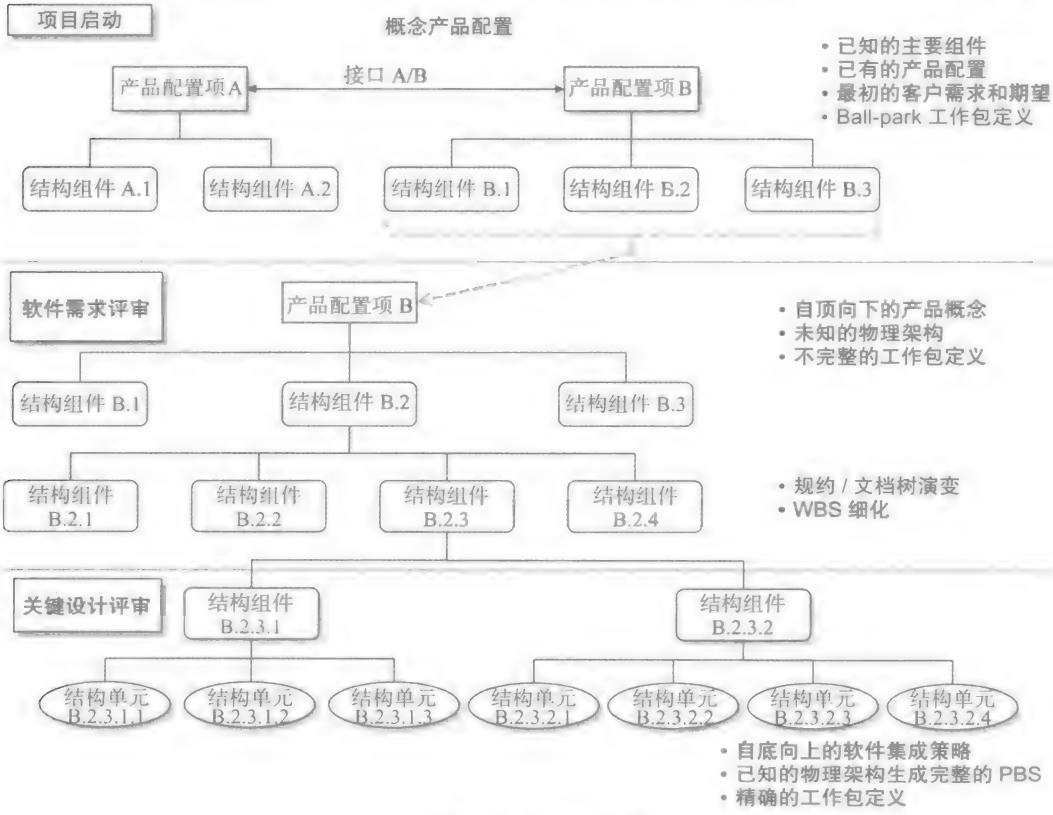


图 4-5　PBS 的演变

架构的改进和特征化而进行的概念性演变。当分析了软件需求、研究了可选的设计方案并选择了设计方法时，演变就发生了。该图只显示了一个配置项的局部分解。产品分解结构的定义在早期软件定义活动中不断扩大，但是都应该在关键设计评审（CDR）时终止。依据这个里程碑，软件产品物理架构应该由全套可识别结构单元和软件集成策略来完成。

4.3.3　规约树

规约树是一种能够确定需求规约以及在开发过程中它们与软件产品和开发后的过程关系的原理图。产品或过程结构分解的每个元素都应该有一个规定性能或重要特征的规范，这是涉众的需求，也是必须交付的。规约通常以一种能使开发人员和涉众衡量软件产品操作目的或业务目标适用性的方式来编写。规范代表一个输出、一个产品或者一个由工作分解结构确定的工作包。规约树支持开发项目中的成本估算、预算和配置控制活动。

4.3.4　文档树

文档树是一种能够确定在开发过程中与软件产品或开发后的过程相关的设计图纸、图表、文档和技术手册的原理图。产品或过程分解结构的每个元素都应该有一个或更多的记录设计特点的文档项。文档代表一个输出、一个产品或者一个由工作分解结构确定的工作包。文档树支持开发项目中的成本估算、预算和配置控制活动。

65 / 66

4.3.5　软件产品基线

需求基线确定了所需的软件功能、性能和接口特征，必须在将要完成的产品开发中体现。这解决了用于证明具有特定需求的成果的验证基础。需求基线包括软件和计算环境需求以及在验收测试阶段软件测试人员用于验证产品完整性的接口规范。传统的软件需求基线被配置管理学科称为功能基线。然而，该基线只解决了软件产品的功能性，却忽略了由计算环境中元素提供的功能性。由于一个软件产品没有合适的配置计算环境就不能执行，所以引入了术语需求基线。从本质上讲，需求基线包括软件配置项的功能基线（如果有多个软件配置项）和计算环境。

分配基线定义了组成软件产品的配置项，并且确定了如何跨过底层配置项[⊖]来分配软件功能和性能需求。它包括所有从顶层软件配置项向组成功能架构的功能组件和单元分配的功能、性能和接口特征。

产品基线描述了一个软件配置项的所有必要功能和物理特征，并进行了必要测试来证明配置项符合指定需求。产品基线包括软件结构单元和组成物理架构组件的“代码”规范。

4.3.6　需求可追踪性准则

对软件产品和开发后的过程之间的关系以及项目管理和控制机制有一个完整的理解是至关重要的。在需求规约之中进行软件需求追踪以及测试工件都是不够的。为了响应设计决策，配置可选方案，需求变更提议和不断变化的涉众需求，许多软件开发文档工件（规范、图表、计划、时间表、预算、工作包等）都必须关联并且通过一个或多个跟踪工具来跟踪。图 4-6 确定了软件设计、文档、计划和应该包含在需求可追踪性指导计划中的项目控制工件

⊖ 术语配置项通常用来识别集成、可交付的软件产品。一个软件产品可能会有一个或多个软件配置项。软件架构中的每个元素（功能性或结构性组件和单元）代表由分配和产品基线决定的配置元素。

的类型。

67

必须管理一套最低追踪关系来支持软件验证、确认评估以及配置审核。这些都在图 4-6 中用实线表示。

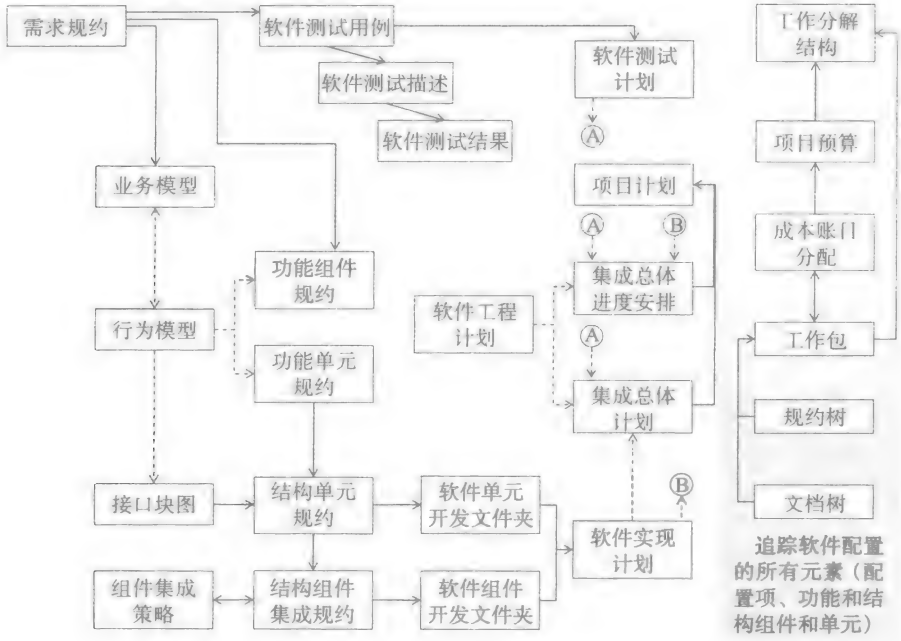


图 4-6　推荐的需求追踪覆盖范围

4.3.7　权衡分析

权衡分析是一种通过评估涉众需求、软件需求或架构设计可选方案来确定优先行动过程的探索性工具。当存在经过考虑的而不是假设的竞争解决方案或者不够理想的设计决策时，就应该进行软件权衡分析。理解每个决策在复杂性、实现解决方案的工作需求，以及在开发后的过程的影响和生命周期成本方面是如何影响软件产品架构是很重要的。每个潜在的可选方案都必须从几个角度进行评估，包括性能、创新性、实现成本、可用性、客户增值、竞争态势和开发后期维护方面。此外，如果选择可选方案，应该通过评估可选方案了解潜在风险来实现项目目标。了解每个可选方案存在的风险可以避免许多风险。

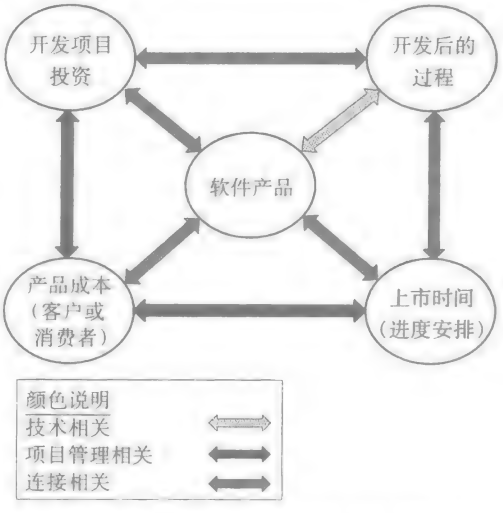


图 4-7　软件开发过程中的基本权衡

在做任何决定来批准软件需求或设计方法之前，有 8 个基本的权衡分析必须要考虑（图 4-7）。权衡分析在影响软件开发或产品成功的主要竞争因素之中产生。这些权衡分析将考虑把开发项目的费用作为一项产品开发投资。上市时间解释了提早发布产品来抢占市场份额并建立客户基础的

重要性。通过产品绩效会认识到，每个功能或性能的增加都会加重工作，也就是提高了产品成本和开发费用。

理解这些产品开发和项目特征之间的关系是很重要的。开发项目的花费表示开发软件产品的成本，然而开发速度解释了考虑到产品投入市场中或交付给客户的时间安排。考虑到产品性能、特征和竞争因素，产品成本暗示了软件产品对于客户或消费者的价值。产品性能解释了软件产品提供的功能和性能特征。包括额外的功能或显著的性能优势在内，由于产品投入市场的延迟导致潜在损失，应该允许产品的价格增加。此外，那些产品特征将会产生额外的开发成本，必须由产品销售来收回。

68
~
69

权衡情况涉及充分理解一个特定选择的上行和下行而做出决策。这意味着丧失一个产品或一种情况的特性或特征而获得另一种。灵敏度和风险分析是两个值得注意的技术，它们支持权衡分析决策。它们在分析结果的不确定性以及这些变化如何归因于不同的假设等方面对可选方案进行评估。

4.3.8　软件复杂性度量

软件工程团队必须通过待评估的软件产品来确定度量措施。至少，这些软件产品复杂性度量措施的建立都应该有助于设计决策。第一个度量应该解决操作或业务流程复杂性[⊖]度量，软件产品的目的是自动化。其他度量措施应该解决与软件产品架构元素之间的相互依赖关系相关的复杂性。

过程的复杂性可以通过构造一个业务或商务环境中静态或动态的软件产品业务模型来获得。软件产品复杂性需要三个相互关联的度量措施来解决产品架构的演变：

1）涉众需求的数量和可感知的计算错综复杂性可以应用于软件需求规格。

2）功能协作依赖关系解释了一些在功能架构之内识别的用户数量、数据库、外部和内部的相互作用或数据交换。

3）产品架构的结构密度解释了组装软件产品所需的一些组件的集成操作。

这三个度量措施为软件产品架构中的每个元素都提供了一个计量。表 4-2、表 4-3 和表 4-4 为计算错综复杂性、功能协作依赖关系和结构密度软件复杂性度量提供了描述。

70

表 4-2　计算错综复杂性度量描述

	名称	描述
度量	计算错综复杂性	软件产品需求带来的全部挑战都关于技术实现中涉及的算法效率、性能、资源利用和其他性能指标
参数	空前的 (U)	对于先前没有已知的解决方案，需求的数量特征化了一项数据处理挑战
	复杂的 (C)	对于已理解的解决方案，需求的数量特征化了一项数据处理挑战，但是实现之后未必会令人满意
	苛刻的 (D)	对于计算机和软件技术不能充分的支持解决方案，需求的数量特征化了一项数据处理挑战
	适中的 (M)	对于已知且相对容易实现的解决方案，需求的数量特征化了一项数据处理挑战

⊖ G. M. Muketha, A.A.A. Ghani, M. H. Selamat, and R. Atan (2008). A Survey of Business Process Complexity Metrics, 可以通过 http://scialert.net/fulltext/?doi=itj.2010.1336.1344&org=11 获取。

（续）

	举例
计算方法	每个需求的分类都是基于感知的难易程度来有效地满足，并且有效地考虑到软件技术、编程语言限制以及与之相关的软件开发团队的经验和技术水平。每个需求的评估都由论题专家来分配一个复杂评级（10 代表广泛；7 代表极度；4 代表重要；1 代表适中）并且整体协作复杂性评级通过将每个分类的评级求和来计算。然后将复杂性评级除以需求总数来得到计算错综复杂性度量的复杂性
公式　(10 × (U)+7 × (C)+4 × (D)+(M))/需求总数	对于这个示例问题，假设软件需求包括以下涉及架构和实现过程中挑战的参数： U=1 需求 意味着得到一个可行的解决方案会遇到空前的挑战。 C=4 需求 意味着得到一个可行的解决方案会遇到复杂的挑战。 D=7 需求 意味着得到一个可行的解决方案会遇到苛刻的挑战。 M=38 需求 意味着得到一个可行的解决方案会遇到适中的挑战。 (10 × 1)+(7 × 4)+(4 × 7)+(38)=104 需求总数 =1+4+7+38=50　104/50=2.08 这个计算复杂性的示例识别出一个中等复杂的软件解决方案
复杂性等级	极度复杂：>5 非常复杂：在 4 和 4.999 之间 高度复杂：在 3 和 3.999 之间 中等复杂：在 2 和 2.999 之间 程序复杂性：在 1 和 1.999 之间

表 4-3　功能协作依赖性复杂性度量的描述

	名称	描述
度量	功能协作	功能元素的安排所带来的全部挑战在于依赖关系和接口
参数	广泛的 (EX)	在一个功能元素（功能组件或单元）之中功能依赖关系或接口的数量超过 6
	极度的 (E)	在一个功能元素（功能组件或单元）之中功能依赖关系或接口的数量是 4 或更多
	显著的 (S)	在一个功能元素（功能组件或单元）之中功能依赖关系或接口的数量是 2 或更多
	适中的 (M)	在一个功能元素（功能组件或单元）之中功能依赖关系或接口的数量少于 2
		举例
计算方法		每个功能元素都是由论题专家进行评估，并且基于功能元素利用功能需求中的依赖关系或接口的数量进行分类。每个分类的数量乘以它的复杂性评级（10 代表广泛的；7 代表极度的；4 代表显著的；1 代表适中的）并且整体协作复杂性评级通过将每个分类的评级求和来计算。然后将协作复杂性评级除以功能元素总数来得到功能协作度量的复杂性
公式	(10 × (EX)+7 × (E)+4 × (S)+(M))/功能元素的总数	对于这个示例问题，假设软件结构配置包括以下涉及结构组件集成的参数： EX=1 功能元素包括超过 6 个功能依赖关系或接口。 E=4 功能元素包括超过 4 个功能依赖关系或接口。 S=7 功能元素包括超过 2 个或更多的功能依赖关系或接口。 M=38 功能元素包括少于 2 个功能依赖关系或接口。 (10 × 1)+(7 × 4)+(4 × 7)+(38)=104

（续）

	举例
公式　(10×(EX)+7×(E)+4×(S)+(M))/功能元素的总数	功能元素总数 =1+4+7+38=50 104/50=2.08 这个计算功能协作的示例识别出一个中等复杂的功能性解决方案
复杂性等级	极度复杂：>5 非常复杂：在 4 和 4.999 之间 高度复杂：在 3 和 3.999 之间 中等复杂：在 2 和 2.999 之间 程序复杂性：在 1 和 1.999 之间

表 4-4　结构密度复杂性度量的描述

	名称	描述
度量	结构密度	软件配置项的结构性安排所带来的全部挑战在于软件集成工作
参数	广泛的 (EX)	涉及 10 个或多个结构元素集成的结构组件的数量
	极度的 (E)	涉及 7 个、8 个或 9 个结构元素集成的结构组件的数量
	显著的 (S)	涉及 4 个、5 个或 6 个结构元素集成的结构组件的数量
	适中的 (M)	涉及 3 个或更少结构元素集成的结构组件的数量
		举例
计算方法		每个结构组件都是由论题专家进行评估，并且基于在集成中涉及集成元素（子组件或单元）的数量进行分类。每个分类的数量乘以它的复杂性评级（10 代表广泛的；7 代表极度的；4 代表显著的；1 代表适中的）并且集成复杂性评级通过将每个分类的评级求和来计算。然后将集成复杂性评级除以结构组件总数来得到结构密度度量的复杂性
公式　(10×(EX)+7×(E)+4×(S)+(M))/结构组件的总数		对于这个示例问题，假设软件结构配置包括以下涉及结构组件集成的参数： EX=1 涉及 10 个或多个结构元素集成的结构组件。 E=4 涉及 7 个、8 个或 9 个结构元素集成的结构组件。 S=7 涉及 4 个、5 个或 6 个结构元素集成的结构组件。 M=38 涉及 3 个或更少结构元素集成的结构组件。 (10×1)+(7×4)+(4×7)+(38)=104 结构组件总数 =1+4+7+38=50 104/50=2.08 这个结构密度的示例识别出一个中等复杂的软件配置
复杂性等级		极度复杂：>5 非常复杂：在 4 和 4.999 之间 高度复杂：在 3 和 3.999 之间 中等复杂：在 2 和 2.999 之间 程序复杂性：在 1 和 1.999 之间

4.4　软件术语注册表

注册表确定了软件产品架构中的架构元素，并提供了关于每个元素和它们与架构中其他元素关系的有意义的信息。注册表提供了附属于注册表中每个条目的配置识别信息。一个术语表应该能够识别功能和结构组件、单元的唯一名称，以确保产品架构定义中不会使用重复的名称。术语表中的名称应该按照字母顺序排列，并且与架构元素相关的应该使用该产品特

有的标识符。图 4-8 描述了一个关于术语注册表的推荐结构。

软件产品
1. 配置项 X
1.1 功能配置
1.1.1 功能组件 X.1
1.1.1.1 功能组件 X.1.1
1.1.1.2 功能组件 X.1.2
1.1.1.3 功能组件 X.1.3
1.1.1.3.1 功能单元 X.1.3.1
1.1.1.3.2 功能单元 X.1.3.2
1.1.2 功能 X.2
1.1.2.1 子功能 X.2.1
1.1.2.2 子功能 X2.2
1.1.3 功能接口
⋮
1.2 物理配置
1.2.1 结构组件 X.1
1.2.1.1 结构组件 X.1.1
1.2.1.1.1 结构单元 X.1.1.1
1.2.1.1.2 结构单元 X.1.1.2
1.2.1.2 结构组件 X.1.2
1.2.1.2.1 结构单元 X.1.2.1
1.2.1.2.2 结构单元 X.1.2.2
1.2.1.2.3 结构单元 X.1.2.3
1.2.3 结构接口
1.2.4 外部接口
2. 配置项 Y
⋮
3. 术语名称

图 4-8　术语注册表的推荐结构

4.5　软件集成策略

软件集成策略是在准备软件产品物理架构时发展起来的。它确定了结构单元将如何组成一套结构组件。结构单元是由功能架构通过对常见的功能单元分组并同化单元规约而派生出来的，用来消除冲突、重复和不一致。然后结构组件通过合成分组的结构单元并同化它们的规约来建立结构组件规约。结构组件规约不应该重申结构单元需求，但是应该解决那些独特的行为特征和集成后的接口。这种对于集成的软件组件的行为特征的理解，来源于将低层结构元素合成为集成结构组件。这种集成策略可能涉及对非开发项的集成，例如商用软件组件或可重用软件组件。

71 ~ 74

软件集成策略为软件实现团队的工作计划来执行组件集成和测试形成了基础。因此，结构组件的标识必须形成软件配置项和产品的组装结构。每个软件组件必须包括足够的功能和测试存根以便可以独立地测试和评估。

4.6　项目和技术方案

软件开发项目的执行是由一套项目和技术方案来指导的。然而，规划和决策是项目成功的核心。方案本身只能证明实现即时或短期目标的行为设计方法或过程。为软件开发项目建立的最初方案需要根据软件需求和确定的特征化架构进行修改。在开发的早期阶段，建立一个全面的方案来解决整个软件开发工作是不可能的。因此，在每个评审或里程碑对项目和技术方案优先进行修改来反映前一开发阶段获得的理解是必要的。图 4-9 展示了软件架构在前三个开发阶段的演变，以及如何改进项目和技术方案。

在软件开发工作开始的时候，项目方案应该简单地建立要实现的主要里程碑以及产生用于定义软件需求的技术活动、文档和模型。在进行软件需求评审之前，应该扩展项目和技术方案来解决用于描述产品功能和物理架构的技术活动。这些技术方案在集成总体方案中进行了总结，并在集成总体进度安排中基于时间进行安排。集成总体进度安排是一种联网的、多层次的进度安排，显示了建立软件产品架构所需的重要任务。这些项目和技术方案应该在软

件需求评审中进行基本元素评审。软件需求评审的成功完成将授权由这些规划设备链接的工作。纵观软件开发工作，应该修改这些方案来调整技术工作，通过剩余工作来完成项目目标。

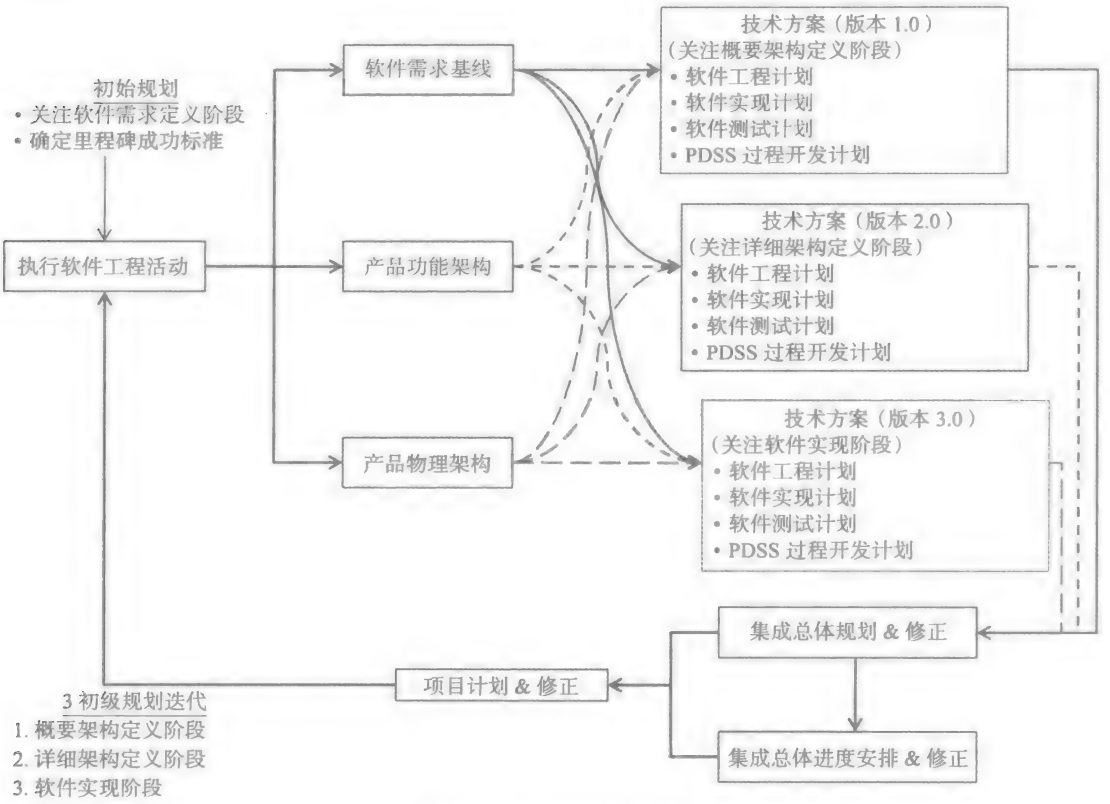


图 4-9　技术和项目规划

4.6.1　技术组织规划

75
~
76

软件工程规划是指导软件开发工作的核心技术管理文档。它关注的是需要定义并控制软件架构的集成技术工作。它应该确定组织角色和责任、集成产品团队的目标和组成，以及阶段性成果和成功标准。

每个技术组织都应该准备自己的个人工作计划，用来确定如何完成工作任务。技术领导负责每个任务并与其他确定能够有助于任务执行的技术组织进行交互。软件实现计划确定了如何设计、编码和测试结构单元，以及如何执行组件集成策略。软件测试计划解释了如何执行包括预测试在内的产品验收测试，以确保软件产品已经准备好进行正式的验收测试。应该在预测试的过程中解决未发现的软件产品缺陷，并且在进行测试准备评审之前重复测试。开发后的过程的开发计划应该解释设计、实现并且测试基于软件产品架构每个开发后期维护过程的方法。

4.6.2　项目规划

集成总体规划和集成总体进度安排表示解决了待执行技术工作的项目规划。集成总体

规划提供了一个关于重要技术工作（活动、高层任务、评审和里程碑）的集成的、分层的意见。集成总体规划不是日期驱动的规划，但是却以必须要实现的技术成果为特征。每个工作都必须通过显式成果和与被用来确认实现预期成果的相关标准来定义。集成总体进度安排是由集成总体规划派生而来的，为实现技术成果的必要任务提供额外的详细内容。集成总体进度安排列出了预计的开始日期、持续时间和任务与其他任务之间的关系，提供了一个集成任务网络。集成总体进度安排基于日历，为项目的执行、监控和进度控制提供了一个关于任务必要细节的视图。集成总体进度安排必须直接追踪集成总体规划，并且必须与工作分解结构关联。

77
/
78

第5章

Software Engineering: Architecture-Driven Software Development

软件集成产品和过程开发

集成产品和过程开发（IPPD）是软件工程的主旋律，反映了产品架构和维护流程。IPPD的目标是以一种降低复杂性和生命周期成本的方式来控制产品定义。IPPD定义了系统开发过程中两种普遍存在的关系：第一，并行工程，它用来保证在设计过程中整个产品的生命周期都是考虑在内的；第二，所有的产品准则，包括所有的实施、集成、测试和评估、维护和支持，都应该包含在早期设计阶段。本章将要讨论IPPD应如何改变以适合软件工程成果。IPPD的成功实施体现在：

- 缩短了投入市场时间；
- 减少了产品开发成本；
- 降低了开发风险；
- 提高了产品质量；
- 降低了维护成本。

传统的IPPD定位了产品设计工程和产品性能之间的紧密关系。涉及硬件元素的产品必须专注于生产和装配线的性能，来产生大量的具有一致维度、属性或者特征的单元。工程公差必须指出允许产品质量中对集成组件的运作没有显著影响的不完善性和固有易变性的合理余地。因此，硬件部分的工程设计说明书必须包含生产加工性能。此外，生产工艺必须包含足够的过程控制装置，来保证认可部分的产出，并且减少制造的浪费和返工。这就要求产品的设计工程师一定要知道生产工具和制造工艺的性能。

一个软件产品是一个由许多密不可分的部分组合而成的复杂的“系统”。简单分解软件的需求和把它们分配给软件组件和单元是不够的。软件IPPD必须通过建立产品架构来解决设计难题和复杂性。功能架构的开发处理了软件控制流、数据交换、性能、资源利用率以及失败状况的检测和响应。物理架构处理了软件结构单元和组件的协作方式，并且被集成来组成最终软件产品。软件架构代表了软件产品的设计，包含了说明结构单元和组件行为的足够信息。

软件IPPD与传统的软件开发不同，因为软件产品的性质——实现设计取决于编译语言固有的功能和编程结构。软件的功能和物理架构必须在明白实施和测试挑战的前提下进行开发。这需要软件工程活动中的软件实现和测试主题专家的参与。另外，必须提出软件的开发后期维护过程，这样它们能及时被建立和测试以支持软件产品分布。因此，软件工程集成产品团队必须包括来自所有软件技术组织的代表。这允许以一种对软件实现和维护问题欣赏的态度来开发软件产品架构。这个方法减少了生命周期成本，把实现项目进度的风险降到最低，也限制了需求变动的潜力。

另外，软件工程IPPD保证创建产品架构花费了足够的时间，该架构中早期设计的决策对项目和过程的成功具有最重大的影响。在架构定义阶段，应该通过建模和创建原型评估设计选择，以确定最好的方法来进行。直到软件实现之前，原型是不应该被舍弃的，因为它将

重要的设计决策推迟到了开发的下一阶段。软件实现应该对照着物理架构建立的规范，简化为软件单元的设计。这个方法与硬件开发更一致，硬件开发在详细设计阶段开发详细模型和设备原型，以验证设计的适用性。

80

表 5-1 列出了传统软件开发，软件工程和硬件开发阶段来突出这些方法的不同。需要注意的是，软件实现和软件产品的设计、编码、集成和测试跟硬件装配阶段是一致的。这种一致性表明软件实现工作跟硬件制造也是很相似的。因此，软件产品设计工作的优势，包括模型和原型，应该提前完成以引导软件实现。

表 5-1　硬件和软件开发阶段的比较

硬件开发阶段	传统软件开发阶段	软件工程开发阶段
HWCI[1] 概要设计——评估分配给硬件配置项的系统需求，制定需求说明书	CSCI[2] 概要设计——评估分配给软件配置项的系统需求，指定软件部分的说明书（软件需求的分配）	软件架构概要设计——评估软件需求，产生软件产品的功能性架构。产生行为模型来评定产品性能。产生功能性部件和单元的识别和说明
HWCI 详细设计——开发模型和原型来评估设计的概念和产生制造工程图样和图表	CSCI 详细设计——评估软件组件的需求，制定软件单元说明（软件需求的分配）	软件架构详细设计——评估软件功能单元，创建软件产品的物理架构。产生模型和原型来支持设计权衡分析。产生结构单元和集成组件的识别和说明。建立软件的集成策略
HWCI 制造——生产机构利用工程图样并产生测试的工作原型	CSCI 编码、测试和整合——基于单元说明来设计、编码、测试软件单元。进行软件组件的整合和测试	软件实现——设计、编码和测试软件单元。基于软件集成策略执行软件集成和测试。进行产品测试来保证软件产品做好了验收测试的准备
HWCI 测试——基于需求说明书正式测试硬件项目	CSCI 测试——基于需求说明书正式测试软件配置项	验收测试——基于需求说明，正规测试软件产品

① HWCI = hardware configuration item，硬件配置项。

② CSCI = computer software configuration item，计算机软件配置项。

5.1　IPPD 在软件中的应用

集成产品和过程开发是以自从 20 世纪 90 年代早期就开始为系统工程体系服务的基本指导方针和假设为基础的。IPPD 的指导准则已经被大部分联邦政府的机构和开发大型、复杂产品的商业组织所接受。IPPD 的基本准则是由国防部长首次建立，他授权 IPPD 在所有系统取得项目中使用。[⊖]

IPPD 被定义为一个集成所有活动，从产品概念到产品 / 领域支持，用多功能团队来同时优化产品及其生产和维护过程来满足成本和性能目标的管理过程。它的核心准则如下：

1）客户至上：IPPD 的基本目标是更好、更快、更实惠地满足客户的需求。客户的需求应该决定产品的性质和它的相关过程。

2）产品和过程的并行开发：过程应该和它们所支持的产品并行开发。关键是，过程用来管理、开发、生产、校验、测试、实施、运作、维护、培训人员，以及在产品开发过程中要考虑的产品的最终处置。产品和过程的设计与性能应该保持平衡。

3）早期和持续的生命周期规划：产品及过程的规划应该在科学和技术阶段尽早开始（尤

⊖ DoD Guide to Integrated Product and Process Development, Version 1.0, Feb. 5, 1996.

其是高等开发），并在产品的生命周期里拓展。早期生命周期规划，包括客户、功能和供货方，为产品及其过程的各个阶段打下了坚实的基础。应当定义关键程序事件，以便资源应用以及资源约束的影响能更好地被理解和处理。

4）最大化承包商独特方法的优化和使用灵活性：提案需求（RFP）和合同应该为承包商的独有流程、商业规格说明、标准和业务的优化及使用提供最大限度的灵活性。他们也应该适应在需求方面的改变并激励承包商来质疑需求并提出成本效益高的可选择解决方案。

5）鼓励鲁棒设计和提高过程能力：鼓励先进的设计和制造技术的使用，它们能促进通过对制造工艺中对变动不敏感的设计和产品获取高质量（鲁棒设计），并致力于过程能力和不断的过程改进。

81
82

6）事件驱动调度：应该创建一个调度框架来将项目事件和它们的相关成就及成就准则联系起来。只有当与事件相关联的成就都按照成就准则完成的时候才能认为完成了该事件。该事件驱动调度通过确保产品和进程成熟度在接下来的行动之前逐步显示而降低了风险。

7）多部门团队协作：多部门团队协作对于产品及其进程的集成和并行开发是必不可少的。需要合适的人在合适的场合、合适的时间做出及时的决定。团队的决定应该建立在结合整个团队（例如，工程、生产、测试、后勤、金融管理、承包人员）的投入的基础上，包括客户和供应商。每个团队成员需要理解他或她的角色，要支持其他团队成员的角色，也要理解与其他团队成员工作的限制。在团队内及团队间的交流应该开放，成功的团队应该强调和嘉奖。

8）授权：决定应该与风险相称，趋于最低可能的水平。应该将资源分配成不同等级，与人们的权限、职责和能力相一致。团队应该拥有与团队能力相称的管理他们的产品及其风险的权限、职责和资源。团队应该承担责任，并为他们的工作成果负责。

9）无缝管理工具：应该建立一个在各级上都将产品和过程相关联的框架，展示依赖性和相互关系。应该建立一个单一的在产品全生命周期里联系需求、计划、资源分配、执行和项目追踪的管理系统。这个集成方法有益于确保团队拥有所有可用的信息，由此来加强团队在各层次上做的决策。通过得到和支持数据库的使用及为使用、交换、查看信息的软件工具，能力应该被证明是在产品整个生命周期里分享技术的和商业的信息。

10）风险的主动识别和管理：与系统特性相关的关键的成本、进度和技术参数应该从风险分析和用户需求角度来确定。应该以恰当的指标指定技术和业务性能评估计划，并将其与同类行业中最好的基准相比较，以提供继续的技术和业务参数的预期和实际成就程度的验证。

尽管这些准则集成了项目管理和技术能力，IPPD 在软件开发中的适用性也必须重视与软件工程相关的技术挑战。在本书中讨论的软件工程实践传递一种 IPPD 方法。因此，下面的部分将讨论 IPPD 这种方式是如何被应用到软件产品的工程中的。

83

5.1.1　客户至上

IPPD 这种方式确定了 4 个主题来完成高质量的软件产品：

1）客户需求应该决定软件产品和进程的特征；

2）产品应该更好地满足客户的需求（提高质量）；

3）产品应该更快地满足客户的需求（投入市场时间）；

4）产品应该能以低成本满足用户需求（降低产品和生命周期成本）；

这些主题都是极好的目标，但是并不容易实现。如果软件开发的方法论或方法足以完成

这些任务，那么就没有对提高软件开发的进一步研究或者修饰的必要了。然而，根据 15 年前 CHAOS[⊖]的报道，软件开发项目的成功率在 28% 左右徘徊，平均 37% 的项目被认为是受到挑战的[⊜]，剩下的 35% 是有缺陷的[⊜]。因此，我们可以总结如下，目前的软件开发方法、工具和技术是不足以达到 IPPD 的信条的。本材料中，这些以客户为中心的主题已成为软件工程哲学的核心。

在整个软件工程过程中，验证和确认的重点是确保软件产品架构的定义满足涉众的需求和期望。术语涉众用来代表技术工作的所有客户，包括项目经理、客户、最终用户、供应者以及产品维护机构。在产品需求、产品架构、维护流程和项目任务之间得到平衡的思想是由权衡分析和风险评估行为支持的。

5.1.2　产品和进程的并行开发

软件开发项目的总范围必须包括软件产品的开发，还有该产品被实现（相当于制造业的软件）、测试和维护的过程。除此之外，一个软件产品还需要设置一个运行的计算环境。计算环境包括一系列的计算设备、工作站、存储设备、网络设备、目标操作系统、中间件和相关应用（比如数据库管理系统，DBMS）。 84

在软件开发项目中有很多被软件产品架构的定义影响的过程。在产品开发中相关的三个主要过程是软件实现、计算环境实现和软件测试。与产品维护相关的三个基本过程是产品分配、产品培训和产品支持。产品支持能更细致地分解成几个过程，如客户支持或帮助桌面操作、问题解答和产品增强。然而，软件开发项目识别这些过程，它们必须与软件产品开发一起被定义、设计、实现和测试。图 5-1 用软件开发框架标出了软件产品开发各阶段，并列出了这些过程开发的工作。

图 5-1　产品和进程的并行开发 85

每个软件进程都与可用的设备、器材、人员、工序和相关资源有关，来支持执行每个分任务。这些进程领域必须以一种及时的方法被定义、设计、实施和测试来支持软件产品的开发调度。这些进程的定义受软件产品定义的影响，并且不能独立实施。因此，受 IPPD 的理论影响的任务是这些进程必须和软件产品同时定义。软件产品设计方法可能对这些进程和生命周期成本中的一个或多个有重要影响。因此，这些进程领域的代表必须与定义软件产品架构的软件工程活动相关。这合作的安排将会在本章后面“多部门团队协作”部分进一步讨论。

⊖ CHAOS，斯坦迪什团队论文，1995。

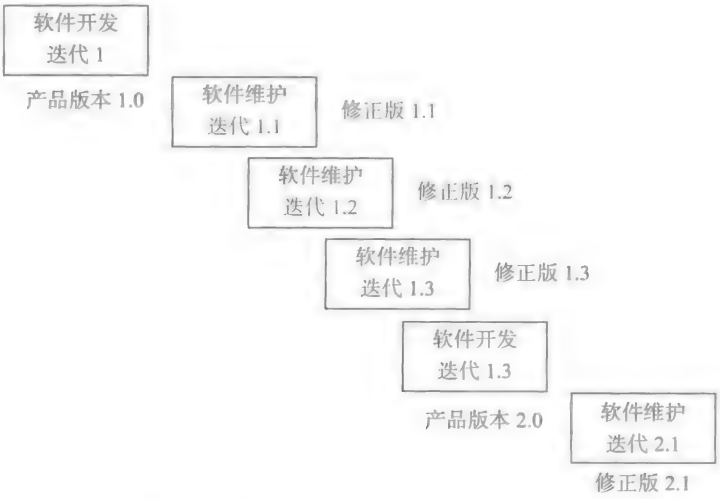
⊜ 该项目已完成并运作但是超过预算、超出预估的时间，而且提供的特征和功能比最初指定的少很多。

⊜ 该项目在开发周期的某个节点已取消。

5.1.3　早期的和连续的生命周期规划

IPPD 理念扩展了开发的定义来处理产品生命周期。这涉及的不仅仅是产品的开发——为开发后的软件维护创建基础设施。这还包括考虑产品经过一系列的迭代开发，可能会怎样随时间演变，还有计算技术的提高会怎样影响产品的设计和计算环境的布置。必须尽早识别出产品生命周期的变化元素，以确保构建软件产品架构来促进产品的演变进化。规划产品开发和生命周期维护行为必须开始于产品概念定义早期。规划活动必须不断修订，以不断地融入完成初步产品开发和启动开发后期工作所需的最新的知识和对工作的理解。

这抓住了迭代和增量软件开发方法的前提。人们认识到一个软件产品在其生命周期中不断发展。因此，围绕软件开发的几次迭代规划软件产品的演化来提供产品功能和性能的增加，是明智的。在 IPPD 理念中，由于建立产品架构的需要，软件产品最初的开发体现了大量的开发工作。软件维护期间，产品被修改、增强，或者被额外的开发行为的迭代所扩展。当软件产品必须经过重大的架构重新定义时，新的软件开发项目应该被创建。对软件开发的每一个主要迭代，相关的生命周期进程可能会需要重建，以适应产品架构的变化。图 5-2 给出了软件开发项目和开发后期软件维护迭代的序列比对。



86

图 5-2　迭代和增量软件生命周期规划

5.1.4　最大化承包商独特方法的优化和使用灵活性

IPPD 这种方式首要关注的是客户、开发者、供应商和分包商之间的协议。创新的技术对于软件开发可能有利也可能对项目成功有害。虽然新的或未经证实的方法可能是有利的，但是还是应该在实验的情况下选择性采用它们，以确保产生预期的结果。

该方式的第二个声明提出了对需求的质疑以及提供可选择的高成本效益方案。这是一个有价值的想法，不管什么时候被应用，都会提供一些益处。质疑需求的有效性提高了软件需求捕获和表示的精确性。涉众代表们通常通过他们自己的观点来看软件产品。因此，他们会支持那些修饰他们所关心的软件产品特征的需求。涉众必须面临的挑战是确定出什么是软件产品必须做的，而不是过分强调那些漂亮、精致或者引人注目的特征的重要性。

软件需求必须从许多资源中征求和收集，必须阐明并按重要性排列所有的需求，以建立

产品功能、特征和其他特性的优先级，每一个需求都产生开发工作的成本，因此质疑协助淘汰不必要或者次要涉众的需求和期望的有效性，目的肯定是减少需求集合到产生一个支持客户和涉众需求和期望的可行产品最小的需求，这是明确规定的，以便不存在涉及需求语句的目的的和在项目资源的范围和计划里程碑相适应的困惑。

在一些例子中，涉众确实分享与他们的特殊需求的辩护相关的经济负担。假如一个公司在投资一个软件开发工作，公司的代表会有用他们的钱得到功能最丰富的产品的责任感。他们可能会歪曲特征的重要性，可能会跟软件开发团队争论不必要需求的价值。这些代表们没有意识到他们用过多的需求加重了开发工作的负担，会导致了开发工作从一开始就注定失败。质疑软件需求的有效性，在决定适合开发项目约束条件下的最小需求集之前对它们排列优先级，这些总是必要的。

一旦软件开发工作开始，任何被提出的需求的改变都会立即受到挑战。对现有需求的新添加或者更改可能会需要重大返工来吸收对产品架构设计的改变。需求变更相关的开销应该包含吸收所有受影响产品文档的设计修改的工作。此外，组织计划、技术规划、工作包、日 87
程表和相关的计划文档必须被更新以应对每次变化。如果一个提出的改变可以被延迟，并被包含在软件产品的将来版本中，那么接受改变可能才是最好的，而不是延缓合并到软件产品中。

5.1.5　鼓励鲁棒设计，提高过程能力

这种方式致力于提高软件设计（软件工程）、实现、测试和开发后的过程衔接。旨在创建促进软件实施（设计、编码、集成）和测试的软件设计技术。该方式对认同软件架构作为软件实现的基础来建立。从软件产品架构设计到软件实施的转变通过为每个结构单元和组件开发软件产品规范得到加强。软件集成策略是在详细架构定义活动阶段开发的，并为软件集成和测试提供蓝图。因此，此处的资料旨在通过提高软件产品设计和过程开发来实现这种方式。

5.1.6　事件驱动进度

不管什么时候提到进度，都是指集成总体规划和进度安排（IMP/IMS）。IMP 是事件驱动进度，标识了必须完成的成就和需考虑完成的工作必须满足的准则。IMS 是时间驱动进度，它把有组织和技术方面的计划与项目里程碑相结合。这在第 4 章中有提到，并根据需要在本书其他部分进行了进一步讨论。

5.1.7　多部门团队协作

执行本书中讲解的软件工程实践需要用到 IPT。主要的集成产品团队是 SWE-IPT，它主要负责开发软件架构。另外，经鉴定，4 个额外的 IPT 确认能处理软件实施，软件测试和评估，开发后期软件维护流程和项目控制。每个 IPT 应该包含这样的成员，他们拥护涉众的需求，还有各技术组织的代表的需求。其他的 IPT 由项目或 SWE-IPT 成立，认为有利于实现项目目标。

5.1.8　授权

软件开发计划和进度活动包含所有的技术组织，为完成通过工作包分配给他们的任务，

这些机构预备了详细的计划。这些计划由 SWE-IPT 整合并组成一个集成技术计划和进度
88 安排（ITP/ITS）。ITP 和 ITS 与其他项目管理的工作计划和维护组织结合，来组成 IMP 和 IMS。工作分解结构、IMP 和 IMS 为任务执行提供了基础，每个技术组织必须为获取结果负责。

专业说来，软件产品架构为软件实施建立了结构单元和构件说明。这些规范允许软件实施团队成员用完整的设计决定权来设计、编码和测试软件单元。软件集成策略为软件构件集成和测试的方法，软件实施团队应该被授权建立软件集成和测试程序。

5.1.9　无缝管理工具

本书为建立集成软件工程工具提供了基础，包括提出计划、产品架构定义、进程跟踪、状态报告、配置控制、扩展需求可溯性和风险管理能力。一个集成工具环境能提供一个多部门的、协同工作的工具框架和数据仓库。

5.1.10　风险的主动识别和管理

风险的识别和管理作为软件分析的一个重要元素，在第 14 章中进行了充分讲解。风险是指任何潜在的、能不期而遇的、对项目目标的达成产生负面影响的所有事物。既然计划和设计需要做一些决定，最好就是确定每个选择固有的风险。这就使 SWE-IPT 在掌握了更多的关于假定的固有风险的知识之后，做架构设计的决定。鼓励风险削减计划来确定将会启动每个意外应变行动的风险跟踪程序和标准。

5.2　软件工程和开发

在第 6 章中对当前形势面临的软件开发项目做了评估。每个软件开发项目的成功都取决于作为基础的产品架构的建立，基于该架构，软件产品在它的生命周期中实施（以编程方式设计、编码、组合和测试）、测试和维护。本书的其余部分将会讲述软件工程和开发任务。软件工程任务在第二部分会讲述。在这一部分，每一章都会集中讲述一个图 5-3 中确定的软件工程流程的主要元素。表 5-2 确定了讲述软件工程过程中的每一个元素的章节。第三部分中的每一章确定了在软件开发的某阶段必须执行的有组织的任务，包括软件工程、计算环境的定义、软件实施、软件测试和评估及开发后期软件维护。

表 5-2　软件工程实践的章节号

软件工程元素	章节号和章名
需求分析	7——理解软件需求
	8——软件需求分析实践
	9——软件需求管理
功能分析和配置应用设计综合	10——制定功能架构
	11——功能分析和分配实践
	12——物理架构配置
	13——软件设计综合实践
软件分析、控制、验证和确认	14——软件分析实践
	15——软件验证和确认实践
	16——软件控制实践

软件开发项目必须围绕 IPPD 原理和方式组织。为实现这个目标，软件开发工作的范围必须识别和处理软件产品对计算环境的依赖性，以及软件生命周期中“生产”阶段的缺乏。软件产品是分布的一种形式，这里电子分布文件或者被复制、打包然后传给分销商或零售商，或者这些文件通过网络服务器提供下载服务。因此，软件 IPPD 组织框架在执行时必须反映图 5-4 中所确定的结构。带阴影的方框代表应该利用 IPT 管理的单个项目元素。这些 IPT 中的每一个都应为他们所管理的软件产品或进程的定义、实施、质量负责。

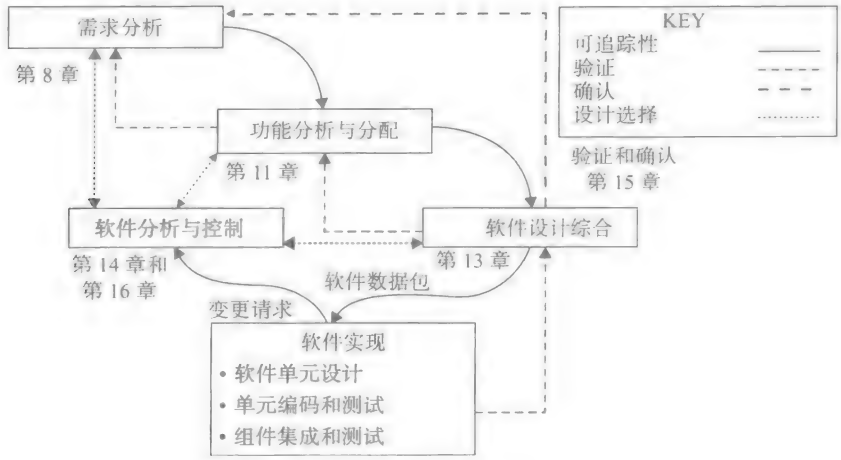


图 5-3　软件工程实践

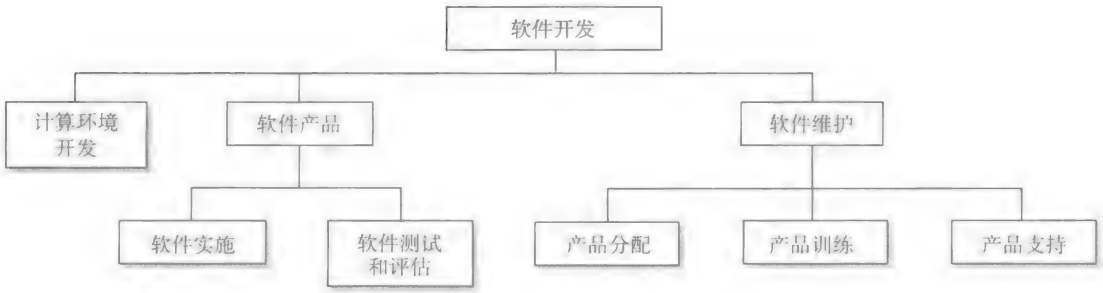


图 5-4　软件 IPPD 组织框架

89
~
92

第 6 章

Software Engineering: Architecture-Driven Software Development

软件设计阻碍

本章研究一个与软件产品开发相关的挑战。研究主要集中在识别软件的固有特征，这些特征阻碍了软件产品设计和用于影响软件设计的非正式的做法。我们讨论作为“原材料”的软件特征，这有助于形成软件产品的挑战。这项探索研究了自从查尔斯巴贝奇在 1837 年设计出第一台叫作分析引擎的可编程计算机以来编程的演变历史。我们审查软件开发的历史来暴露威胁到每个软件开发项目的挑战。软件工程的价值是用事实证明它如何提供一个渐进的、遵守规则的、有益的软件开发方法。

1995 年，斯坦迪什（Standish）小组发表了从 1994 年的第一个 CHAOS 报告，它评估了整个软件开发相关行业的成功率。这个报告描绘了软件开发的状态是极度存在缺陷的。斯坦迪什小组的研究表明，高达 31% 的委托软件项目在完成之前就失败或者被取消了，53% 的项目成本比原估计超支 189%。美国公司和政府机构为取消的软件项目花费了 8100 万美元，而且为完成超过预算的项目多花费 5900 万美元。对当时的美国和世界经济增长来说，软件开发成为一种负累。

自 1994 年，CHAOS 研究每两年进行一次。2009 年出版的 2008 年的报告显示了软件开发成功率达到 32% 的增长，尽管 44% 的项目是有缺陷的（推迟、超过预算和（或）少于规定的特征和功能）。不过软件开发项目的失败率（在完成或交付之前被取消以及从未使用的[⊖]）在 15 年的报告期中，平均为 20%。这表明目前的软件开发实践是不能可靠地按时或在预算内交付软件产品的。表 6-1 提供了 CHAOS 报告结果对 15 年来已收集数据的总结。

表 6-1　CHAOS 报告总结

	1994	1996	1998	2000	2002	2004	2006	2008
成功的	16%	27%	26%	28%	34%	29%	35%	32%
有缺陷的	53%	33%	46%	49%	51%	53%	46%	44%
失败的	31%	40%	28%	23%	15%	18%	19%	24%

造成这种混乱状况的原因，是软件作为材料缺乏物理性质，因此传统工程实践不能应用到软件产品的开发。软件从业者没有建立软件工程学科的任何技术基础。软件行业的开拓者们设计了无数的软件开发方法，它们在软件开发成功上几乎没有让人认可的提高。只有一个行业中 30% 的成功率被认为是可观的，没有人会期望一个棒球运动员始终打平均 0.350 以上的成绩。

软件已经成为众多消费者产品的决定性元素。这成了一个重要问题，因为软件纳入了公众、政府机构、公众和私人机构每天必须依赖的系统中。软件义务和消费者保护的问题让关注重点回到目前软件开发的方法、技术和潮流的不足。软件产业必须建立正规的软件工程实践，基于这种软件工程实践，软件开发能够演变成一个可靠的行业，并且能显著提高软件开

⊖ 参见 *http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html*.

发项目成功率和产品可靠性。图 6-1 给出了从最初的实验室试验到消费的软件开发趋向的概念进展。软件编程语言和设计技术的发展与证明这些技术如何支持软件开发作为一种合法职业的发展阶段是相关联的。

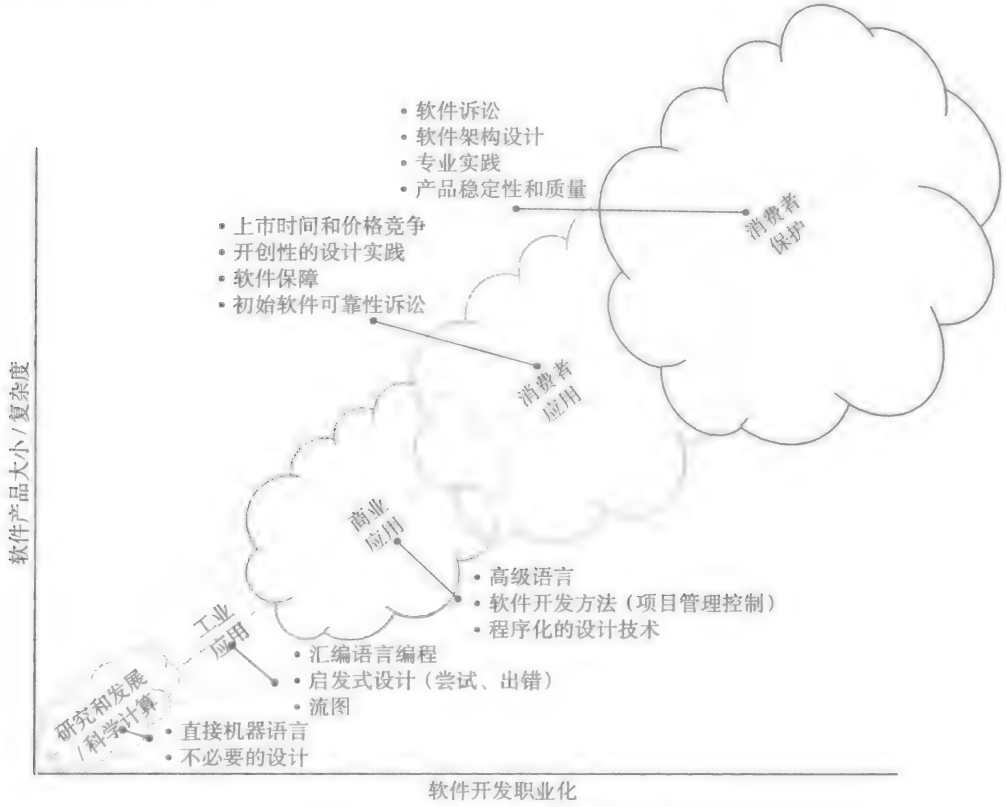


图 6-1　软件开发技术的演变和扩散

图 6-1 所示的过程代表了大部分新技术的典型进化路径。消费者保护法和保护机构的存在是为了确保产品不会因为引进新技术而造成严重的伤害、损害和财产损失。将软件引入关键系统和消费市场会激励软件工艺向专业地位的必然过渡。当前的软件开发工具、技术和实践必须由实际工程的实践、政策和过程来加强。

已经有相当多的研究是完成的将一些准则应用到软件产品的开发。接下来的各节将通过审查软件开发技术、发展和管理实践的演变，来回顾困扰软件行业的状况。

6.1　作为原材料的软件

工程是应用科学、数学和技术来设计人为的建筑物、机器和其他人造产品。这涉及原材料或零件被制造、组装、集成或整合，以形成更大、更复杂的产品。软件作为原材料，没有表现出任何指导软件产品设计的科学特性。软件从根本上是一系列程序，程序由可以转变成机器可读格式的语言的指令组成。在计算机处理器级，这些指令被编码为正负电荷，表示二进制值（0 和 1 分别代表接通和断开状态）。计算机处理器执行各种基本的数学计算来转换数据值。计算机操作系统管理计算的命令执行和软件产品与计算机输入 / 输出设备和数据存储设备的交互。

计算机科学是一个为推动编程语言和计算理论的进步提供了研究和发展动力的研究领域。计算复杂性理论调查数据进程算法是高度抽象的基本属性，而计算机科学的其他分支，如计算机图形学，强调现实世界的应用。编程的研究调查了软件语言的定义和结构，以解决复杂的数据处理事务。计算机科学的研究往往交叉其他学科，如语言学、数学、物理、统计和逻辑。然而，对于为软件产品设计或工程建立一个可靠的方法，只有微不足道的关注。大多数研究都被降级到低一级的方案设计启发式。百科全书词典定义启发式为“解决不存在公式的问题的方法，基于非正式的方法或经验，并采用试错迭代的形式。”

软件产品与早期的编程语言的开发开始于简单问题求解启发式。这种方法由用于描述执行的计算函数或程序所需要的数据处理操作的顺序的流程图技术做补充。作为一种技术，流程图对确定控制机制（决策块和控制流程）、输入/输出过程、数据处理步骤（例如，$X = X+1$）、调用子程序（通常详述在一个单独的流程图）和数据并发处理是有用的。由于描述控制流任意跳转的 goto 语句的不良使用，采用流程图失宠于 20 世纪 80 年代初。这导致“意大利面条”代码，使得理解数据处理流程变得困难并影响软件维护的挑战。然而，使用 goto 语句的是纯粹非结构化编程或不当的编程设计技术的结果。

软件是一个广泛的术语，它用于识别各种操作计算机和通过计算机技术嵌入的设备的计算机程序。术语硬件用来描述一个系统或基于计算机的系统的物理元件，而软件是指用于操作基于计算机的系统的各种程序或应用。软件代码是能被转换成计算机可执行格式的一系列指令。代码被分为程序、子程序、函数、模块、对象或者形成更大程序的部分或元件的其他构件。由于计算机程序不断增加的规模和复杂性的挑战，数据处理业务的这种划分开始了。

在软件领域，有各种术语的引用，必须明确这些引用以区分软件的一些基本属性。
96 表 6-2 给出了这些软件产品相关术语的大体解释。

表 6-2　关键软件术语和定义

术语	定义
代码	以一种编程语言的语法表达，忽略代码的正确性和数据处理结果的有效性可以被编译和执行的指令
模块	一个早期的结构化编程术语，来指代自包含的例程或程序，表示一个较大程序的一部分或一个元件。模块代表了关注点分离，通过被称为接口的强制执行的界限提高软件可维护性。在一个较大的程序中，模块通过模块接口的调用被执行。模块接口表示调用的时候被模块提供和返回的数据项
模块性	一个软件程序的各部分的划分和相互关系。模块编程甚至可用于编程语言缺乏支持命名模块的语言特征的情况
对象	在面向对象编程中，一个类就是一个模块，它封装数据属性和用于设置和检索它们的值的一套程序。一个对象是拥有一个唯一标识的类的实例（具体的值使它区别于这个类的其他对象），是描述存储在对象中数据值和指定使用对象的接口的状态
程序	执行一系列数据处理任务业务或操作的模块或对象的组合。程序的执行形式是二元形式，这样计算机可以执行。人类可读形式是以一种程序设计者可以编辑的软件语言在源代码中表示。源代码形式的程序必须被编译、汇编、链接和其他必要的引用库例程来生成一个可执行文件
应用程序	设计用来执行业务或操作任务的特定程序。应用程序是一个用于区别通用程序（比如文字处理、电子表格、视频播放器等）和系统软件与中间件的术语。应用程序管理并整合计算机的功能，但并不直接支持用户执行业务或可执行的任务
系统软件和中间件	中间件是为软件应用（除了那些可从操作系统获得的软件应用之外）提供计算服务的软件，或为软件应用之间提供联系以便它们交换数据的软件
产品	任何正在开发的商业分配、客户交付或者为促进事业发展的目的明确的软件程序或应用。用于处理对产品的软件开发成果的专注和其开发后期工作的通用术语

97

软件应该被视为一系列促进业务或操作流程的数据加工事务。数据加工事务描述了软件执行的刺激反应性质。几乎所有的软件模块都是由一些刺激发起，执行一些计算操作或函数来产生一些结果，传输数据处理控制给另一个事务。一个大型的软件程序应该包含若干可能的事务，这些事务是由一系列模块或例程的执行来完成的。模块执行的顺序依赖于计算活动的输出，和一些形式的决定事务如何推进的决定和控制逻辑。

因为缺乏物理特征，软件表现为一种人工语言形式，用于：

- 进行数学计算或将符号转换成有意义的数据；
- 建立引导数据加工事务流的逻辑推理；
- 为以后的访问以数字形式保存数据；
- 与计算环境元素交互；
- 与业务或操作系统元素交互。

因此，软件是语言学、语义学（意义的研究）、数学符号、逻辑、知识表示和系统工程（复杂问题处理）的组合。

由于软件产品规模越来越大，复杂性越来越高，系统工程实践因其强调分析设计问题和降低设计复杂性变得越来越适用于软件开发。然而，还没有权威的指导原则来激励或鼓励软件行业采用系统工程规范。软件专家们提出无数的方法，旨在提供更好的软件开发策略。然而，大多数软件方法学都由快速发展的计算机技术或编程语言（如面向对象编程语言）所驱动。其结果是，大部分的软件从业者缺乏必需的基本技能，以应付软件产品设计相关的错综复杂的困境。

6.2　软件技术的变革

软件领域的研究仍然处于细化的早期阶段。它涉及一些努力跟上计算机技术发展的研究主题。另外，软件技术的应用领域迅速蔓延到社会的几乎每一个方面，包括交通运输（飞机、船舶、汽车、卡车、火车和交通管理系统）、通信、娱乐、商务信息处理、医疗卫生、建筑、制造业、公用事业、批发 / 零售、金融服务（银行和投资）、教育、个人计算等。然而，软件产品的编程和设计技术的发展已远远落后于对软件日益增长的需求。 98

通过观看软件相关技术的时间表，可以更好地理解作为一门技术的软件进步。图 6-2 描述了 70 年来部分计算机技术和编程语言演化的概貌。[⊖]这张图强调了那些促进了软件产品开发的优秀编程语言。1945 年和 1985 年之间，与软件编程相关的亮点见表 6-3。[⊖]程序设计开始作为一种专门的技术由实验室技术人员来努力推进计算系统。这些科学家最初曾用机器代码（二进制 1 和 0）在非常大的有真空管的计算机上为非常简单的计算编写程序。机器代码是一组指令，由计算机的中央处理单元（CPU）直接执行。

引入 C++ 以后，大部分编程领域的贡献都涉及对已有语言的扩展，来适应面向对象编程或者基于网络的应用开发。20 世纪 90 年代早期开始引进集成开发环境（IDE），如微软的 Visual Studio。IDE 为软件开发提供了一组集成的工具，如源代码编辑器、图形用户界面（GUI）构造器、软件构件库、调试器和构建自动化工具。软件编程语言的演变似乎已经按常规发展，因为重点已经转移到提高程序员生产力。

⊖ 参见 *http://en.wikipedia.org/wiki/Timeline_of_computing* 和 *http://en.wikipedia.org/wiki/Timeline_of_programming_languages*.

⊖ 参见 *http://www.computerhistory.org/timeline/?category=sl*.

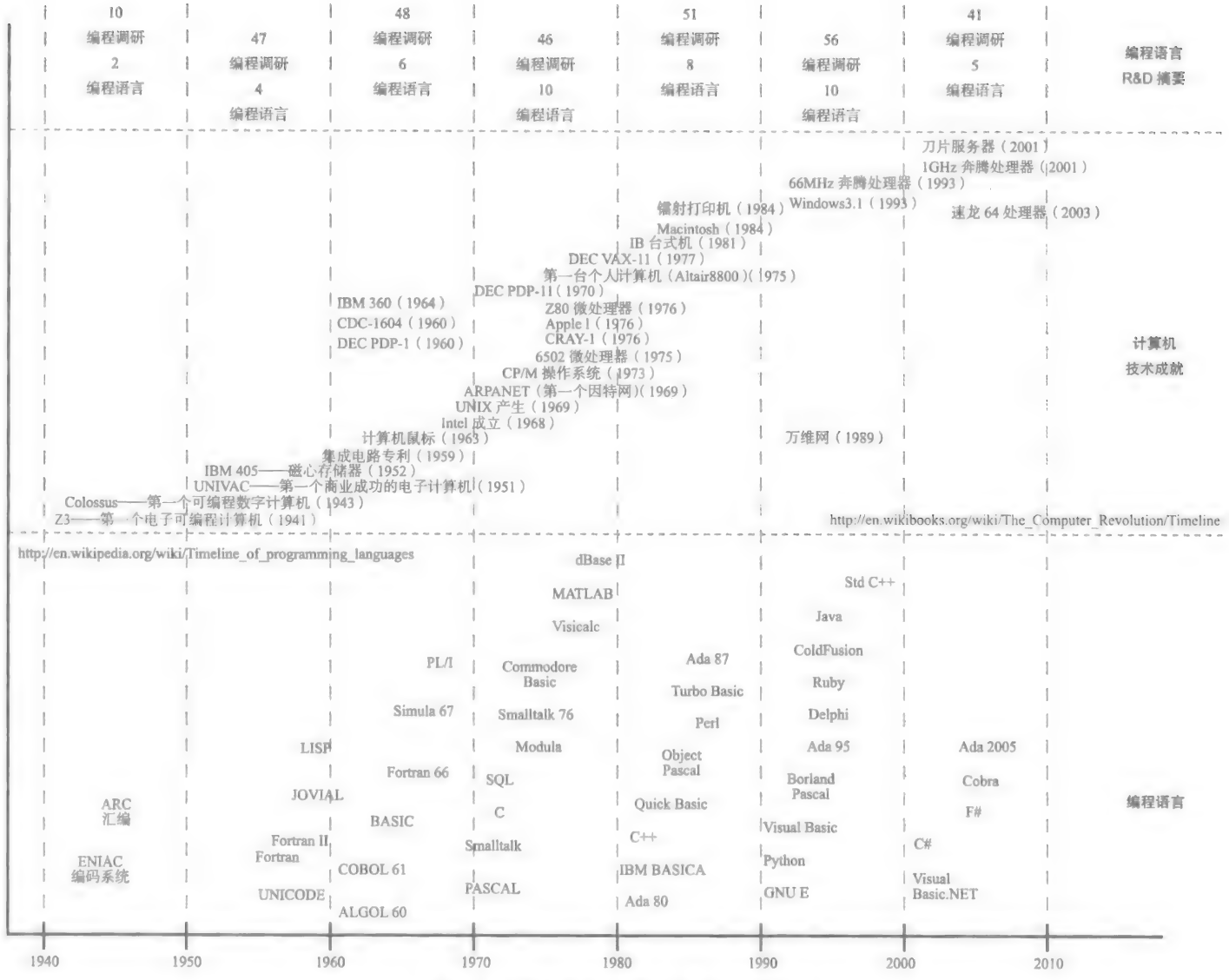


图 6-2 部分计算机和编程技术的时间线

表 6-3　软件程序语言里程碑

时间	软件里程碑
1945	Konrad Zuse 开发了 Plankalkul（Plan Calculus）——第一个算法编程语言
1948	Claude Shannon 的《通信的数学理论》讲解了工程师如何编写数据，以便在计算机之间传输后他们能检查其准确性。Shannon 确定了比特为数据的基本单位，同时，也是计算的基本单位
1952	数学家 Grace Hopper 完成了 A-0 编译器，它被认为是第一个编译器，一个允许计算机使用类似英语的单词而不是数字的程序
1953	John Backus 为 IBM 的 701 计算机完成速度编码。虽然速度编码要求更多的内存和计算时间，但它削减了数周的编程时间
1957	FORTRAN（FORmula TRANslator 的简称），使一台计算机通过使用循环执行一组指令的重复任务
1960	来自计算机制造商和五角大楼的团队开发了 COBOL（common business oriented language，面向商业的通用语言）。为商业使用而设计，早期 COBOL 的努力目标为计算机程序的易读性和尽可能多的机器独立性
1962	Kennetn Iverson 出版了他的书——《一门编程语言（APL）》，这就产生了第一个实际的编程语言。APL 在科学、金融，特别是精算应用中广为使用。APL 中的强大功能和运算符用特殊字符表示，产生非常简明的程序
1963	ASCII——美国标准信息交换码，允许来自不同制造商的机器进行数据交换。ASCII 由 128 个 1 和 0 的唯一的字符串组成。每个序列代表英文字母表中的一个字母、一个阿拉伯数字、一个标点符号的分类标记以及符号或函数，如回车
1964	Thomas Kurtz 和 John Kemeny 为达特茅斯学院的学生创建 BASIC——一种易于学习的编程语言
1965	Kristen Nygaard 和 Ole-John Dale 写到，通过模拟面向对象的语言得到了一个早期的提升。模拟将数据和指令分组到称为对象的块，每个代表系统用于模拟的一个方面
1969	AT & T 贝尔实验室的程序员 Kenneth Thompson 和 Dennis Ritchie 开发的 UNIX 操作系统上的一个备用 DEC 小型机。UNIX 结合许多分时和由 Multics 提供的文件管理特征，并因此而得名。（Multics，20 世纪 60 年代中期的项目，创建多用户、多任务操作系统的第一次努力。）UNIX 操作系统迅速获得了广泛的追随，尤其是工程师和科学家
1976	Gary Kildall 开发了 CP / M——一个个人计算机的操作系统。CP/ M 的广泛使用使得程序的一个版本在由 8 位微处理器构成的各种计算机上运行成为可能
1979	哈佛大学的 MBA 候选人 Daniel Bricklin 和程序员 Robert Frankston 开发了 VisiCalc，这使个人计算机成为商务机，用于苹果 II。软件 VisiCalc（visible calculator）使电子表格的重新计算自动化。一个巨大的成功是每年售出超过 10 万份
1981	出现了 MS-DOS（Microsoft disk operating system，微软磁盘操作系统）——新发布的 IBM PC 的基本软件
1985	C ++ 编程语言成为了计算机行业的占主导地位的面向对象的语言，当 Bjarne Stroustrup 出版《C++ 编程语言》一书的时候。Stroustrup 在 AT & T 贝尔实验室说，他的动力源于编写事件驱动模拟的渴望，需要一种比 Simula 更快的语言

在 1960 ～ 1990 年的 30 年间，编程语言迅速变化的情景抢占了学生、教师和从业者的注意力，因为他们努力保持他们熟练和有销路的技术。新的编程语言出现，旧的编程语言朝着融合大幅增加计算能力的方向演化。计算平台、语言、存储技术、计算机图形化以及多媒体技术的改变已经对软件行业的产品生产能力形成挑战，其可靠性和持久性满足了客户、消费者、企业管理者或投资者的需求。

6.2.1　软件开发方法和标准

软件产品的开发，已经通过一系列的编程语言、计算机技术和软件方法支配的形式演变了。早期的软件程序是很小的，是程序、子程序、函数或者模块的不复杂的组合。流程图表技术最初用来提供设计基础，在该基础上软件代码可以产生，转换为机器可执行格式（编译和汇编），并且验证正确执行。当软件程序的大小和复杂程度变得更大，并有更多的行业特

性时，它们被称为软件应用。然而，流程图技术不能相应地表达这些更大的设计难题。

一系列渐进的软件方法或设计技术自 20 世纪 60 年代以来就进步了，并试图提高软件开发成功率。图 6-3 提供了一个时间表，展现了各种开发方法、技术和标准的出现。[⊖]此图显示了 12 个关注软件工程或软件产品设计的主题，5 个与软件工程不相关的主题，3 个与编程自动化相关的主题和 7 个记录了软件开发最佳实践的标准。每个这些类别的主题在表 6-4 中有简要表述。

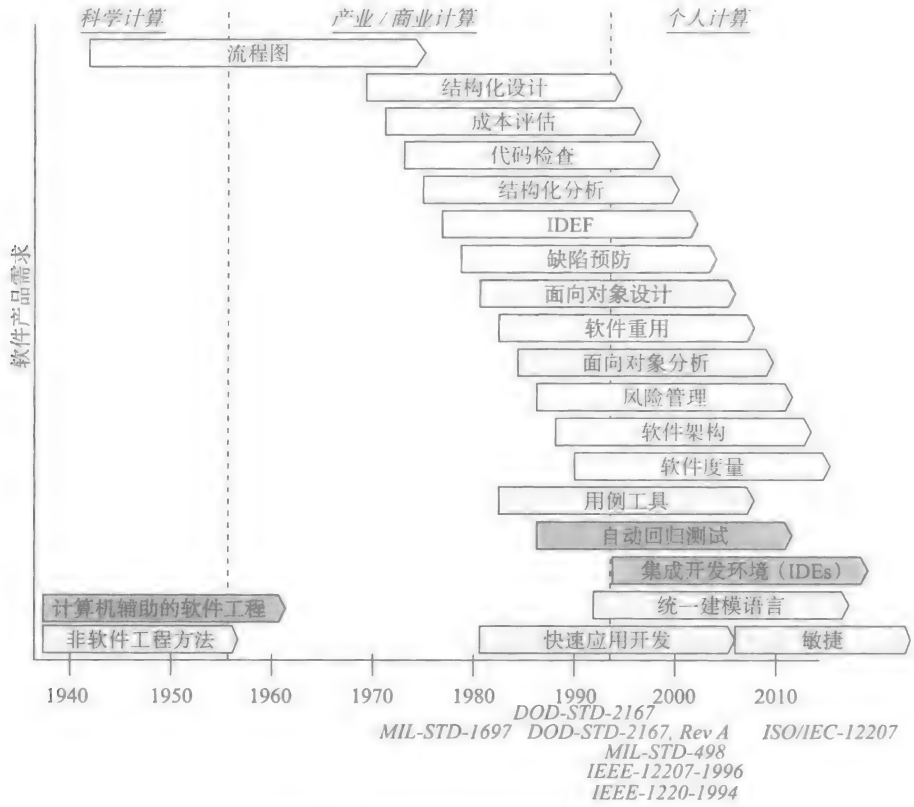


图 6-3　软件开发方法和标准的进展

表 6-4　分类软件开发主题

主题范围	具体项目	详述
软件工程相关	流程图 结构化设计 成本评估 代码检查 结构化分析 缺陷预防 软件重用 风险管理 软件架构 软件度量	这些主题代表了适用于软件工程领域，但不构成一个独立的技术用于开发软件产品架构。有几个对于软件工程来说并不是唯一的，但是是典型的项目管理实践

⊖ 数据提取与定义来自 David F. Rico, "Short History of Software Methods."

（续）

主题范围	具体项目	详述
软件工程不相关	面向对象的设计 面向对象的分析 敏捷 统一建模语言 快速应用开发（RAD）	这些主题中的3个与面向对象编程语言的实现相关。RAD和敏捷是非结构化团队管理方法，专注于快速原型设计和短期规划
自动工具	用例工具 自动回归测试 集成开发环境	自动化工具支持软件实现，尤其是代码生成和测试
软件开发标准	MIL-STD-1679（海军） DOD-STD-2167 DOD-STD-2167，版本A MIL-STD-498 IEEE-12207-1996 IEEE-1220（系统工程） ISO-IEC-12207	解决软件开发活动和任务的工业和国防部标准，重点是软件项目管理、文档和配置控制。注意：IEEE-1220代表了最明确的系统工程标准原则和实践

尽管所有软件行业为严格管制软件产品的开发付出了努力，但实践的状态依旧混乱。软件开发方法需要严格的设计技术来支撑，该设计技术能够与从原料中开发出软件产品的相关困难抗衡。开发任何产品都需要对一套科学的准则和实践的有纪律地坚持，它们正是为了使原材料制成人造的部件和组件而建立的。组装和集成这些部件成为更大、更复杂的组件；然后测试这些部件、组件和最终产品。在制造业这个过程被称为构造、组装、整合和测试（FAIT）。此外，有一个对所有产品开发工作的研究要素，以力求明白新的产品的基础需求。产品设计的这种探索和分析元素为产品、组件、部件和FAIT策略建立了说明规范。这个调查元素负责产品、组件和产生该产品的部件的说明、分析和合成（SAS）。

102
/
103

软件工程实践为完成SAS和开发FAIT策略提供了框架，这就产生了一个完整的、不复杂的软件产品架构。软件需求分析实践将涉众的需求和期望转化为软件产品、结构组件和单元说明。功能分析提供了系统技术，以理解为完成已经分解到软件功能的产品需求，该软件产品必须执行的数据处理事务。软件设计综合实践建立软件产品的结构配置和FAIT策略。结果是一个完整的软件产品体系架构，记录为软件技术数据包（TDP）。软件TDP包括说明书、图表、制图和促进软件实现（方案设计、编码、集成和测试）的软件集成策略。

软件行业一直不能或不愿意加强从混乱的技术到一个训练有素的工程专业的转变的要求。软件专业人士对采取更严格的一套做法来开发软件产品的不情愿，是由于他们对其他工程学科的无知和缺乏经验。无知是福是一句老话！这意味着，由于知道或接受事实的不愉快或先兆后果而宁愿不知道一些事。这不是责备或侮辱软件专业人士；无知仅仅是一种意识、知识或教育的缺乏，或者禁止他们追求更好的方法来设计软件产品。因为软件人员没有意识到还有一种更好的方式来履行他们的使命，他们继续尽力抓住新提出的战略希望隐藏他们能力的缺乏。大多数软件专业人员已经被培训来编写程序，这主要是一个低层次的设计手段。还没有这样的软件方法，它提供了一个全面的方法来设计一个完整的软件产品。

软件原型方法作为一种无须花费太多力量设计完整产品、快速开发软件产品的方法，其出现顺理成章。这种方法已经以各种秘密的尝试被重命名和重包装，来将注意力从正规设计方法的明显缺陷转移，并允许软件专家执行他们知道如何去做的操作，也就是程序。软件原

型的概念起源于命名为快速应用软件开发（RAD）的软件开发方法。RAD 法表明，软件产品可以这样被开发，“最小规划和原型的增量构建。利用 RAD 开发的软件的规划与软件本身的编写是交织的。没有广泛的预案通常会使软件写得更快，并使更改需求更加容易。”[⊖]这导致了敏捷宣言的建立，它试图使软件原型正式成为一种真正的软件开发方法。

从根本上讲，软件开发工作只有当一系列指导其教学、执行与管理的准则和实践被建
104 立时才能提升到专业地位。不允许软件开发方法一再成为新风尚，这是避免软件开发行业混沌状态的原因。软件开发缺乏对设计产品的关注。大部分方法能正确认识到需求很重要和代码生成很容易。然而，对于建立一个严谨细致的完整软件产品的创建设计的方法，一直没有显著的贡献。方案设计方法是在设计层次的编码级别，因此，完全不足以建立一个软件产品架构。

本书中的术语架构是用来区别大部分工程学科中所指的设计。软件产品表现为有助于数据处理事务的各种子程序的组合。没有既定的在结构上整理和组织这些软件程序的设计指导原则。这对产生新的软件产品是一项基本挑战，是软件缺少物质特征的结果。系统工程的准则和实践提供最相关的学科，基于它来建立软件工程的能力。系统工程提供严密的方法来为复杂的产品建立架构。因此，源自系统工程的软件工程实践必须被升为软件工程专业的基础指导方针。

6.2.2　敏捷宣言

敏捷宣言的提出者肯定曾困扰于他们发现自己对于自己职业的情况。没有软件开发方法、技术或使他们成功的实践。因此，敏捷宣言的支持者们构想出一个宣言，来建立一个围绕基于快速模型、增量产品交付和绝对没有产品设计的软件开发的一系列准则的公会。他们需要说服主管：存在一种软件产品能交付的方式，提供“价值”给他们的客户。为达到这个，他们想出一种牵制的战术，如果被接受了，就使他们能做他们已经训练的——程序。图 6-4 介绍了敏捷宣言，为讨论其价值和隐藏错误表示提供依据。

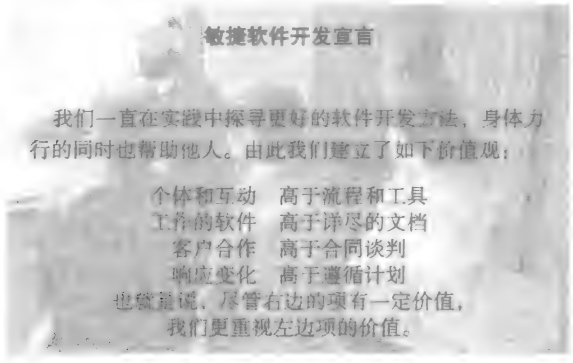


图 6-4　敏捷宣言[⊖]

宣言以提倡敏捷软件开发的公告开始，“在实践上探寻更好的软件开发方法。”表面上，这句话是说通过开发软件的小行为，设计方法和技术自然得到提高。然而，软件开发技术 40 多年的历史没有产生这样的感知。软件开发行业成功的记录低于 30%，并且尽管采用了

⊖ 参见 *http://en.wikipedia.org/wiki/Rapid_application_development.*

⊖ 参见 *http://agilemanifesto.org/principles.html.*

敏捷技术，也没有显著提高。因此，对于公认的软件开发艺术提高的声明，还缺少证明。

敏捷宣言确定了倡导者欣赏的信念。这些信仰是他们在过去的软件开发工作中的经验总结，并解释如下：

1）个体和互动高于过程与工具。软件人员和他们一起工作的方式，对于软件开发工作的成功来说，比过程或工具更重要。这种说法没有基础依据，因为大多数专业的职业已经得出结论，过程和工具对成功、成本控制和质量结果是至关重要的。

2）工作的软件高于详尽的文档。该声明暗示敏捷是基于快速原型方法产生工作原型。该原型演变和改进，以提供最终的、可交付的软件产品。轻视软件文档暗示记录软件设计对软件产品开发没有贡献。如果软件设计没有记录，那么怎么让程序员的团队理解？这表明，软件开发最好通过软件原型的开发，软件原型是不被设计或文档记录的。

3）客户合作高于合同谈判。在ISO标准中，在软件开发过程中有一个过程是建立指导软件开发工作的协议。该协议可能是一个正式的商业实体合同或管理和软件开发项目之间实体的本质。然而，合同或协议用来保持软件开发团队对自己的行为负责。敏捷宣言的支持者不想被追究责任。（难道不是每一位员工都享受不用责任的报酬？敏捷宣言是每一个玩忽职守者梦想的工作协议！） 105 ~ 106

4）响应改变高于遵循计划。变化是不可避免的，但肯定有一个可以衡量进展的计划。通过表明软件产品可以没有计划地生产是虚伪的。每一件产品开发项目必须有一个计划，可衡量进展和使开发团队负责。哦，这是正确的！因为敏捷宣言，软件开发组织不承担责任。那么，一个计划的好处是什么？设计的定义是建立东西的详细计划。除此之外，随着客户不断变化他们的期望，软件开发团队可以享受长时间的参与和持续的就业，因为他们不对进度延误负责。（什么时候产品完成？谁在乎！）

敏捷宣言的作者和支持者肯定没打算要这样公然和明显地解释敏捷的指导原则。他们显然已经放弃了试图遵循标准的项目管理和软件开发技术，由于在被这些实践活动拖累的同时造成的大量令人失望的结果。他们没有意识到将产生实现软件开发计划、目标或协议的任何方法。他们确定，他们成功的最好机会是做他们一直训练做的项目。这就产生了一个宣言，一个当项目失败的时候废除计划、进度表、设计和文档、协议或罪责的技术准则。

当一对舞者进行无情的舞蹈训练以求赢得华尔兹比赛的世界冠军时，他们都致力于自己的专业。当他们的机会来了，音乐开始……这是一个探戈……这对舞者跳华尔兹。毕竟，华尔兹是他们一直以来训练执行的，任何其他反应都会导致失望，因此为什么不跳他们知道如何跳的舞蹈呢？

以前对敏捷的评估必须纳入考虑。敏捷是一种编程方法，作为软件实现的一种形式运作良好。然而，除非它是被软件工程和预定义的软件架构强化，否则敏捷不能成功。敏捷的支持者只是不知道在这个文稿中所描述的软件工程，因此试图“做到他们所能做到最好的”。当我回顾了我对敏捷宣言最初的论述时，我就意识到当它被转移到软件开发的软件实现阶段时他们所提出的价值。当加上前述软件工程工作的时候，敏捷或任何其他软件方法的用处被大大提高了。下一节将讨论一种架构驱动软件开发方法，它结合了软件工程和敏捷方法，将大大提高软件开发项目的成功几率。 107

6.3 架构驱动的软件开发

软件开发的当前状态被大大削弱，因为还没有关于如何开发一个软件产品架构的指导。

千方百计地根据方案设计技术或方法来生产软件产品失败了。即使是那些被认为是成功的努力，也已导致软件产品遭受一个考虑不周的架构的结构。这导致了昂贵的软件维护工作并缩短了软件产品的生命周期。

软件工程提供了技术核心和管理，使软件实施能良好地规划和执行。未在软件工程中受过教育的程序员努力理解涉众的需求，保持行动的战术计划并利用项目资源平衡感知客户的价值。如果一个软件开发项目只通过交付目标驱动，那么确保软件产品交付的机会不大。因此，为什么不直接接受不可避免的需要，逐步实现不完整的产品，希望能安抚客户和管理层。相信客户随着源源不断的增量交付也能收获价值，这是令人欣慰的。然而，客户面对一个在变化的产品配置中再培训的频繁需求，这将最终使客户放弃以往得到一个完整、稳定的产品的希望。虽然产品改进是可取的，但频繁和剧烈的变化只会使产品架构不稳定，并让人产生更多怀疑，最终崩溃、混乱和动荡。

对建立一个可以伴随未来的增强和新特征而随时间演变的最初软件产品版本来说，软件产品架构的需求是极为重要的。软件工程必须解决涉众的困惑、理解问题空间，并建立一个提供稳定而持久的产品基础的结构设计解决方案。历史证明，任何试图开发一个没有架构框架的产品的软件开发工作有 30% 的成功机会。

在这个手稿中描述的架构驱动的软件开发模式如图 6-5 所示。在维恩图的中心是对成功的软件产品开发最重要的软件工程实践。在该模型的顶部是项目管理框架，这是广义的，以解决项目目标、预算、计划和进度表。其余 6 个相交圆圈代表典型的软件开发阶段，包括需求定义、架构定义、软件实现和验收测试。概要设计和详细设计阶段被重新定义来建立基于软件工程实践的软件产品架构。软件技术数据包提供了产品实现所需的规格说明和支持设计信息。

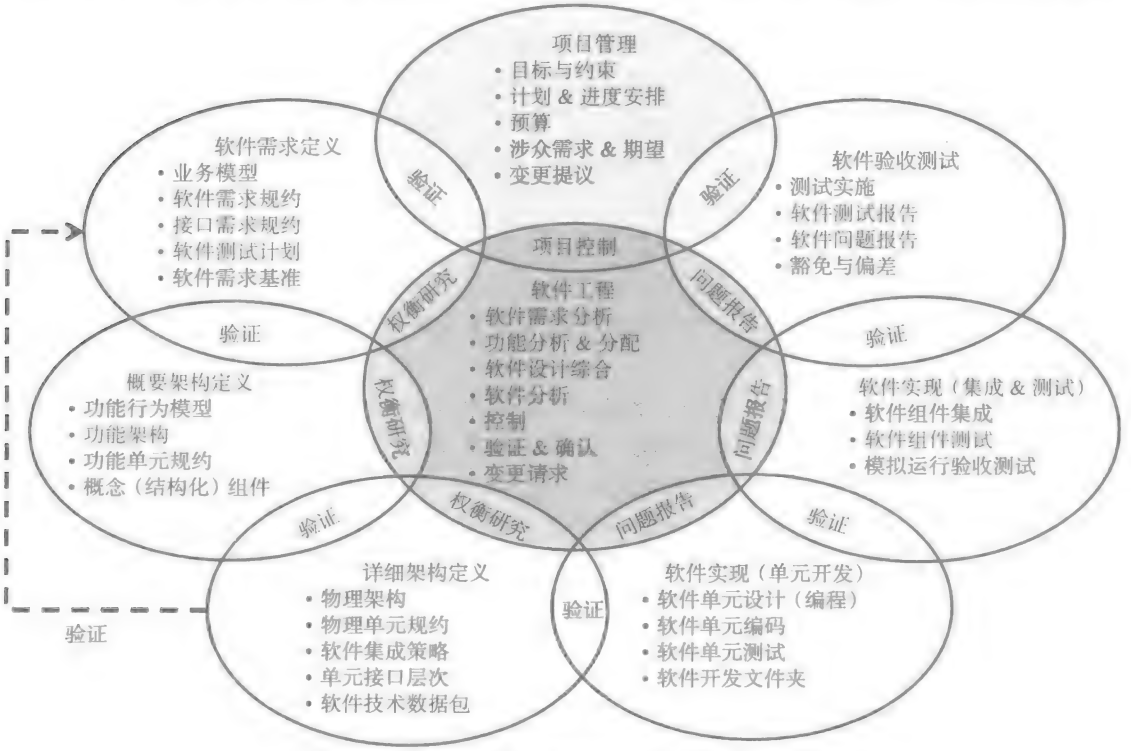


图 6-5　架构驱动的开发模型

这个模型提供了一个软件开发的全面方法，这是基于以软件工程为核心，规划和执行软件开发工作。它涉及的代表来自各种软件学科，以确保软件产品的需求是完整和一致的，在既定的资源预算下技术方案可以实现，软件产品架构提供了贯穿整个产品生命周期的结构上可靠的对软件的改进和演化的框架。

以前的软件方法的失败是由于对如何设计和培育软件产品架构缺乏了解。而软件开发标准和管理实践已经得到了很好的设计，软件方法已经利用编程语言的优势，以使用方便适当的编程设计技术。一直缺乏的是建立一个软件产品架构必要的知识和技能。图 6-6 描绘了一个软件开发的敏捷驱动的模型。这种模型省去了软件工程核心和集中建立软件产品架构的定义阶段。如果没有这个软件工程基础，敏捷只是回避所有重要的“设计”的困境并从需求阶段直接进入软件实现。（注意：根据敏捷宣言，确定项目管理元素的需要是困难的，因此它一直包含在敏捷驱动的模型中。）

108
~
109

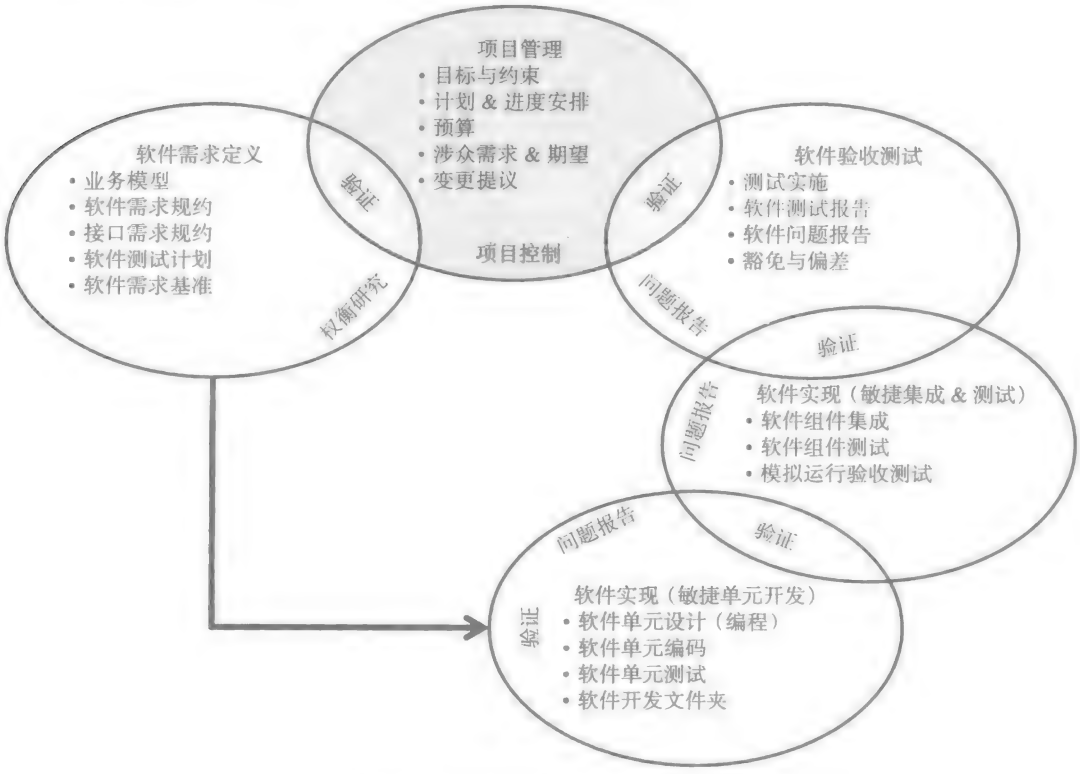


图 6-6　敏捷驱动的软件开发模型

敏捷软件开发模式是由于无法理解软件开发如此困难且易失败的软件从业人员感到失望的产物。因此，敏捷的支持者们建立了软件专业人士公会，他们都相信一个不断发展的产品的增量交付给客户提供了价值。为了“卖”出这种方法，他们把自己与软件开发相关的最有问题的元素隔离开：

- 在编码开始之前导出一组完整的要求。
- 着眼于增量交付与全面的项目计划。
- 无视设计和记录软件产品的需要。

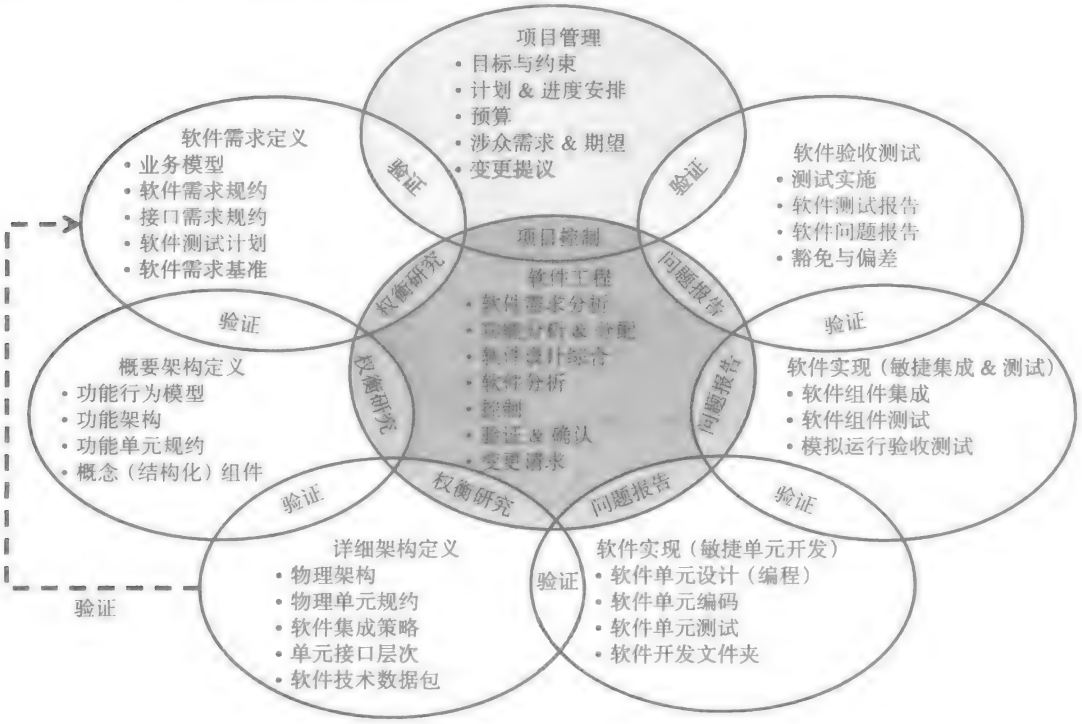
更少的时间浪费在不必要的开销的任务上，而更多的时间花在修复产品上，因此它有

效果。

相反，这些与软件开发相关的问题元素恰好是软件工程的能力。软件工程为使涉众的需求和期望凝聚成一个完整的且一致的软件需求提供了基础。对既定的软件需求基线的变更是
110 对软件架构、项目计划和资源分配的审查，以保证建议的更改能被适应。如果有足够的资源确保项目的成功，必要的变更可以包含入技术计划和进度表。否则，应为未来的调整变化做出安排。

软件架构的建立允许软件实现工作有适当的范围、有计划、有效地实现。物理架构确定的每个结构单元都被充分指定为允许编程级设计、编码和测试。该软件集成策略是在详细的架构定义阶段建立的以允许软件集成和测试，按照既定的工作计划进行。该软件的工作分解结构确定分配给每个软件实现活动的工作包和资源。结果应该是不复杂的，系统的软件实现和完整的软件产品交付测试。

敏捷方法的评估承认如果没有软件架构，软件实施是很难以组织的方式来计划和完成的。软件工程建立了结构基础，在此基础上可以预计软件的实现计划。必须赞赏的是，任何忽略了软件架构的价值的软件方法论或方法都是注定要失败的。图 6-7 描绘了软件开发维恩图，包括软件工程和敏捷方法。此模型可以改变，以适应任何软件实现方法。不管哪里出现“敏捷”这个术语，只需插入首选软件实现方法！



111
~
112

图 6-7　结合软件工程和敏捷的软件开发

第二部分

Software Engineering: Architecture-Driven Software Development

软件工程实践

这部分介绍用于开发软件架构的 6 个软件工程实践。这些实践为所有的工程学科提供基础并为每一个学科独特定制。这些软件工程实践已经应用到了系统工程学科，因为它们处理与控制产品的复杂性。每个实践都表现在一系列有助于探究问题 / 解决方案空间的任务，探索一个切实可行的，负担得起的架构决议。

这些实践并不是独立的，必须根据需要迭代和递归应用，以设计出决定性的解决方案或者进一步约束问题空间来实现可行的解决方案。为了允许问题 / 解决方案空间能以分层的方法探究，我们迭代应用实践。此外，实践必须被迭代应用来重新考虑基于以前定义的架构元素所做的架构决定的含义。软件工程实践的迭代应用就允许问题 / 解决方案空间用自顶向下
113 的结构方法被细化。

这些实践被递归应用到允许深入了解问题 / 解决方案空间并带来影响上层架构解决方案的表面技术挑战。这些实践可能会在继承层次被引用，因为问题被一再探索返回到一个架构的更高层次。实践的每一次迭代应用或递归应用都针对架构问题 / 解决方案空间被认为的目前水平。

每一个软件工程实践都提供与所有专业的工程准则相关的基础结构。这些结构在下表中定义。

软件工程实践	工程结构
需求分析	涉众需求和影响架构解决方案的业务条件。提供一个说明需求的分析方法，能使涉众、开发者和项目管理人员清楚
功能分析与分配	调查这些行为——划分与产品、业务环境以及维护流程相关的功能和性能特征。标识出产品必须准备遭遇的失败的环境，以及支持有效操作的可用资源
设计综合	标识有助于满足指定需求的重要解决方案。得到设计图样和模型，确认设计解决方案达到需求的有效性。建立产品的结构配置并生成“构建”图纸，以及建立将制造的物理元素（部件、组件等）的规范。
软件分析	提供设计供选择的对照的监督管理，来评估潜在解决方案的优点和缺点，以确定采用最平衡的方法。这涉及评估与被提议的架构选择方案相关的风险，来理解采用某一架构方案的潜在的结果
验证和确认	确保描述架构解决方案的三个角度与发展的架构的定义是一致的。这三个角度包括产品需求、功能和物理架构
控制	确保架构设计决策和审定的变化在整个产品构造文档和项目计划中是一贯能透彻理解的。为工程活动的人工制品提供稳定性，以确保它们恰好保管在用于记录目的的受控库之内
114

系统工程实践提供机制来调查架构设计问题并创建一个有效的、高效率的架构方案。这些实践在设计或产品分解的每一级上应用，以确保问题空间被充分理解并且方案考虑到了所有可能出现的潜在操作状况。

这些软件工程实践已经适应于支持软件产品的工程。它们已经根据为达到每个实践目的所执行的各软件工程任务而定义。这些任务由一系列软件具体功能和结构元素构成，这允许读者了解在产品分解中任务应用到了哪里。

在功能架构中，功能组件和单元之间是有区别的。功能组件表示复杂的数据处理功能，即应该被进一步检查来揭示支持功能组件需要的子组件或单元的数据处理功能。当然，有几个（超过两个或三个）层次的功能组件必须要描述甚至适度复杂化软件行为。功能单元表示不需要进一步分解的不复杂的功能。

物理架构中有三个不同的等级，在这之上必然要讨论重要结构元素。在最高等级，概念组件被识别出来代表大型软件产品设计环节的占位符。最低等级包含基础的结构单元，它

们作为物理架构基本的构建块，从功能单元中提取到。结构单元之上的等级是基本的结构组件，这些组件构成逻辑上的或可操作的单元组合，而这些单元组合有助于实现普通的数据处理功能。基本的和概念的结构元素之间的设计分歧是集成组件，其目的是促进软件集成和测试策略。

需求必须针对软件架构的每个元素详细说明。因此，需求分析实践必须理性地应用于架构的每个功能和结构元素。作为一个整体，需求分析实践应该被用于帮助对功能分解的每个层次的研究，以确保问题/解决方案空间被理解。

每当有多重可能的策略，都应该采用软件分析实践，通过这些策略，需求可能被详细说明，复杂的数据处理功能可以被分解，或结构元素可以被分类整理或者集成。

软件验证和确认应该定期进行来确保条理清晰地描述进化的软件架构。这涉及验证三个架构角度是一致的且是一致规定的。确认涉及保证完整的结构组合已经被适当安排来满足软件说明。

软件控制提供了各种不同的任务，来得到完整的架构成品和对不断进化的架构文档的维护控制。这包括对变更需求和建议的处理，以及技术计划、进度表和工作包的更新。 115

另外还有几个章节用来提供详细解说，关注三个架构角度的发展。第7、9、10和12章适于参考之用，不得被视为相关实践的决定性或总结性的描述。这些章节用来帮助读者评估每一个软件工程实践及其应用。

开发软件产品架构

软件工程实践提供一系列的任务来将涉众的需求翻译成一个完整的、一致的、有效的软件产品架构。当实践在架构建立中以连续的步骤使用时，就会出现基本的错误。这些实践不应该被解释为暗示着该架构这样构想出，首先建立需求基准线，然后功能架构，最后物理架构。这个序列意味着瀑布方法，抑制了问题/解决方案空间的适当探索。软件需求分析、功能分析与分配、软件设计综合和软件分析的主要实践，以一种与涉众的需求和期望以及工程资源约束、技术准备和工作人员熟练度均衡的方式，提供了得到完整架构解决方案的基础。

当计划软件工程工作时，有几个演化软件产品架构的概念方法必须考虑。这包括软件工程实践怎样被应用来产生架构。大部分软件方法无法提供迭代特性的设计程序。因此，早期的架构设计决策被假定为最终的并且永远不再重新访问，甚至当它们强加风险来达到项目目标时也如此。流行的软件开发方法（比如迭代、增量、敏捷）已被接受，因为软件开发团队无力掌握完整的架构解决方案。不管方法的选择，每一个软件开发项目都会从软件工程恰当的应用和为软件产品建立一个持久耐用的架构基础得到巨大的好处。

软件架构方法

每一个架构都可以视为一系列软件产品的逐渐详细的描述。架构的层次提供了一种划分问题/解决方案空间的方法。这个概念被比作剥洋葱，在潜入表面到细节的下一层次之前，架构的一层要被考虑。通过逐步引进它们到架构解决方案，为描述涉众产生的架构提供了一个合理的方法。例如，一本书由章组成；章由节组成；节由段落组成；段落由句子组成；句子由单词组成。书旨在讲述一个故事，每一章为下一章提供基础，以这种方式故事朝结尾发展。先看故事是怎样结尾的会使读者错失故事很多宝贵的方面，而这些是为了使故事更加完整、令人满意、持久。 116

开发软件架构为软件产品创建了结构基础。架构的每一个元素都达到一个目的并且其存在的理由是合理的；否则，它就不应该被包含在解决方案中。结构组合或物理架构中每个元素的目的都必须是可追踪到功能架构的元素、指定的软件需求和涉众的需要的。然而，这并不是强制架构解决方案以分层的方法被设计。此外，如果软件架构被允许增量或迭代地进化，随时间流逝，新特性或增强的功能包含进来，软件架构的稳定性和持久性可能会逐渐毁坏。因此，最初的软件架构必须以这样的方式得到，即容纳软件需求到软件产品的第一个版本，同时为产品进化和未来扩展建立基础。

采用关键路径或降低风险分析的架构开发策略可能是谨慎的。关键路径分析承认，软件产品必须设计有一个或更多必要的或关键的数据处理操作。这种方法建立了保证最重要数据处理目标的数据处理方案、行为和结构元素。中等或不太重要的数据处理目标接下来以最高效率、有效和实际的方式，被集成到架构组合。降低风险分析侧重建立围绕分解解决方案最具挑战方面的数据处理操作的架构组合。这提供了一个锥形，来应对最困难和最不确定的设计工作，建立可行的技术解决方案。其余的功能和结构元素可以接着以高效率、有效和实际的方式集成到架构组合中。两种方法都深入探索了限制的架构解决方案，来为整个软件产品建立一个基础的架构框架。

迭代软件工程应用

117 软件工程实践提供了一种类似系统思考的解决问题的战略方法。[⊖]每一个软件工程实践都提供一个不同的、不可避免的且令人信服的解决问题的角度和可用的解决方案。因此，这些实践不应该采用单独作为一组连续任务的隔离。在操作中观察到，每个实践可以引导两个或更多相关实践。一个实践可以被打断来促进其向另一个实践的偏移，根据需要，说明被探索问题或被研究解决方案的某些部分。每一次迭代带着额外的信息返回中断的实践，基于这些信息可以进行进一步分析或综合。当错误推测边缘化时，假设可以被测验和确认。这个迭代为软件工程团队就问题空间提供了明确性，该问题空间产生一个更加有效和高效率的架构解决方案。它给出了一个促进创造力、独出心裁和足智多谋的理解水平。这产生一个解决方案，该方案操作恰当、资源高效、稳定、结构持久耐受。

大部分的软件开发方法都与迭代的概念斗争。它们强调需要遵循结构化的方法，该方法中需求在设计方法完成之前被建立。这促进了典型的需求、设计、编码和测试策略。然而，需要认识到的是，软件产品的需求必须是可实现的。为保证需求的可行性，首选的方法是冒险进入设计领域，甚至是原型，来保证每一个指定的需求都能被满足。随着整个软件开发形式被评估，有一个对立力量的有趣动态，它批准迭代的方法用于任何工程问题。技术团队和涉众之间的互动允许了，对软件操作边界、产品特性和性能挑战、基本结构组件的扩大理解的进化的披露，解决方案也是基于这些被建立。这被称为软件开发的自顶向下或结构分析方法。大部分程序设计者本能的反应是“原型”一个解决方案，并用它跟涉众交往。这被称为软件开发的快速应用开发或敏捷方法。两种方法都没有出现无可争议的成功记录，软件开发行业仍被它尝试建立一个可靠的方法以制订工程准则的无用尝试所阻碍。

⊖ 系统思考已定义为一种解决问题的方法，该方法将问题视作整个系统的一部分，而不是对特定部分、结果或事件进行响应，可能造成意想不到的后果进一步加重。系统思考并非一件事情，而是一系列框架内的习惯或实践，该框架基于这样的信念：一个系统的组件可以组件互相之间或与其他系统关系的上下文中得到充分理解，而不是在孤立状态下。见 *http://en.wikipedia.org/wiki/Systems_thinking*.

在本部分规定的软件工程实践建立了一个迭代范例，容纳软件开发工作中遇到的动态力量。这些潜在的力量包括需要：

1）建立一套可行的软件需求。

2）建立软件实现的坚实的结构基础。

3）与涉众一起增量探索形成的解决方案，来确保双方一直达成共识。

4）确保所得到的产品可根据项目或合同规定交付。

随着软件工程团队试图将解决方案与其驱动需求相结合，出现了一个令人不安的困境，必须再设计一个联合的产品解决方案。必须促进工程解决方案的方法涉及一个同时存在于自顶向下和自底向上方法中的改进。自顶向下架构设想给出了问题空间不间断的阐明，像涉众需要和软件需求中所表达的那样。自底向上实施过程说明了一个产品可以被组装成所需要的构件材料。桥接这两个联合的角度涉及一个二元动力，以转化行为的数据处理线程和为软件组件集成提供策略。 118

这个二元规则导致了一个软件设计鸿沟，这将在第 12 章讲述。软件产品架构以自顶向下的概念的方式逐渐成熟，然而却以自底向上的方式在结构元素的识别和详细说明中努力成为权威。最小的结构部件或单元代表构件块材料，据此软件产品将被实施。每一个结构单元都必须被精确指定，以便软件实施团队可以基于结构单元的详细说明来执行每个软件单元的编程设计、编码和测试。随着较高级的概念组件和较低级的结构元素之间的空间分歧，自然形成了设计的鸿沟。设计鸿沟可以由进一步的自顶向下功能分解被桥接，通过明确自底向上软件集成策略，或者通过两个方向运作。

因此，软件工程实践不是软件产品架构形成过程中的一个独特的步骤。它们是单一的、包罗广泛的问题调查和设计发展范例的元素，被配置来处理无数潜在的软件架构陷阱。由于大量潜在的架构异常和不协调，建立每个软件工程实践的明确描述是不可能的。因此，应该认识到，本部分出现的软件工程实践定义为强调呈现 90% 的完成度和满意度的一组软件工程实现。 119～120

第 7 章

Software Engineering: Architecture-Driven Software Development

理解软件需求

软件开发项目都面临的挑战是满足多个涉众，他们每一个都坚持有偏见的观点，这影响着软件产品的设计。每个涉众代表着与该软件产品相关的一个重要的方面或角色，如产品的性能、测试、对软件实现的评估、产品支持和用户培训。每个涉众群体都把他们与软件产品相关的关心和期望作为重要的特征或特性，必须要包含在软件需求中。

每个涉众所拥护的是不同的，他们经常就那些在软件需求定义过程中必须被（SWE-IPT）处理的目标、观点和期望展开竞争。这种多元化的利益产生了需要权衡研究的行为来决定的设计挑战。表 7-1 确定了每个涉众群体的目标和动机。SWE-IPT 的主要目标是：

表 7-1　涉众的目标和动机

涉众	目标	动机
客户	• 有效软件应用 • 采购成本 • 交付进度表 • 运营成本	• 应用程序功能的扩展性 • 执行的性能和响应 • 计算机环境资源使用
最终用户	• 易用性 • 产品学习曲线 • 敏感的产品支持	• 直观的用户交互 • 有吸引力的、创新的用户接口 • 有效的培训和用户文档 • 客户支持 • 问题的解决和纠正
企业管理	• 客户满意度 • 行业声誉和市场增长 • 项目的投资回报	• 项目的进度和状态 • 成功项目完成的风险 • 项目成本的积累和升级
项目管理	• 项目目标的实现 • 可预测的工作评估 • 项目范围的保持	• 项目规划 • 成本和进度预留
软件实现	• 易于实现 • 产品的可接受性 • 产品质量	• 产品的复杂性 • 不断变化的产品需求 • 规范的完整性和准确性
软件测试和评估	• 产品验收 • 产品质量	• 明确的规格 • 测试环境、工具以及程序 • 计算环境和多用户负载限度
开发后的过程	• 分配流程准备就绪 • 培训计划和辅助设备准备就绪 • 支持流程准备就绪	• 产品包装和分配 • 经销商协议 • 可执行文件复制 • 培训材料 • 问题报告和解决 • 客户支持 • 软件产品增强和扩展 • 产品注册

1）征求涉众的需求和期望。

2）详述产品和开发后的过程的软件需求。

3）生成完成软件开发工作的完整的技术方案。

4）将涉众的需求转化成一个完整的、详细的架构描述。

5）确保软件技术数据包是足够详细的，来促进有效的和高效的软件实现。

6）监控开发工作的进程和开发后的过程的建立。

7）评估提出的更改对软件产品的影响，以确保变更在可用项目资源范围内被容纳。

这些目标大多数都涉及软件需求。软件需求必须针对开发工作的完整范围，包括计算环境、软件产品及其接口，以及开发后的过程。技术规划源自软件需求，且项目团队为实现项目目标的进程根据该技术规划来衡量。变更提议直接涉及可以由任何涉众提出的、对需求基线来说的新事物或修改。变更需求影响产品架构表示的软件产品设计。这种架构源自于软件需求，并由软件数据包确定。为了理解决定项目范围和软件需求的软件产品目的、功能、特征和性能特点，征集涉众的需求和期望是必要的。

最感兴趣的、值得进一步推敲的目标是软件技术数据包。技术数据包是提供给软件实施团队来指导方案设计、编码、集成和测试软件产品的流图、图表和说明的集合。软件实现代表着在住房或系统开发行业的构建或者生产活动。建筑设计师生成一组工程计划和图纸，为施工队建造房子、大楼、桥或其他建筑物提供必要的细节。这包括确定执行建筑规划必要建筑材料的类型和数量的材料清单。在制造业中，工程图纸或图表是提供给生产团队来制造、组装、集成和测试产品的，以确保与图表相符。软件技术数据包包括软件产品实现所必需的架构规范、图样和图表。这包括在实现阶段，确定和详述每个结构单元设计、编码和测试所需的材料的软件清单。此外，它提供了软件集成策略，这描绘了结构单元和组件应怎样组装、集成和测试，直到一个集成的软件产品构造实现。

涉众的需求和期望必须被累积、调解、协调，并划分优先顺序。由此产生的一系列需求必须进行裁剪，以平衡在软件开发项目可用的资源的范围内，满足需求所需要的工作预测。一旦软件需求达成一致，整套软件规范应在配置控制之下定基准和位置。

软件需求为技术工作计划的精炼和细致提供了基础。在 SWE-IPT 采用软件工程的原则和实践为软件开发项目构想技术规划。技术规划始于任务定义、任务依赖和确定工作流的相互关系，以及各个任务和工作流必要的结果。各技术机构的作用必须被确定，且对资源的需求确定。这些任务定义和资源要求被合并到工作包，与项目时间表对齐。基于任务的工作包被组合成工作更大的元素（活动），并组织进入技术工作分解结构（WBS）。WBS 是通过增加项目管理和其他非技术工作包来表示要执行的全部工作，以及各自的预算和进度信息完成的。WBS 提供了准备技术和项目计划的必要细节，如软件工程计划、集成总体规划（IMP）、集成总体进度安排（IMS）和项目计划。

建立软件需求规约以及技术和项目计划需要一个严谨的、纪律严明的方法，涉及权衡分析（技术）、成本－效益分析（管理）和风险评估，以确保需求规约是完整的且能够通过生成的计划、进度表和资源预算来完成。每项任务范围的识别和确定必须能够反映定义、设计、实现软件功能和软件需求中指定特征所做的工作。图 7-1 描述了软件需求是如何建立并反映在技术和项目计划中的。以下各节将描述该工作流的每一步，即将涉众需求转换成需求规约以及现实的技术和项目计划。

图 7-1　软件规格和项目计划的建立

7.1　第 1 步：征求涉众需求与期望

收集、分析和化解涉众的需求和期望以生成软件需求规范。各种涉众会利用业务或业务流程术语或与其特定领域或专业知识相关的表示来表达他们的需求。因为它涉及所需的软件产品，所以各涉众持有不同的观点。有必要收集、分析，并按优先级排列涉众的需求，来开发软件需求规范。涉众需求的优先顺序应该是由每个涉众群体的相对重要性和每个需求的意义决定，因为这涉及产品的验收。通常，如表 7-2 为参与一个典型的软件开发工作的涉众提供了一个相对的优先顺序示例。

注意到很重要一点，软件工程的角色不包含在涉众的列表中。SWE-IPT 负责建立引导软件的开发产品和开发后的过程的软件需求和产品架构。因此，SWE-IPT 必须在其审议、优先级划分和分析涉众的需求中是公平的。这可能是具有挑战性的，因为 SWE-IPT 涉及来自各种涉众组织的代表。

企业涉众代表组织实体，他们为企业批准和监督软件开发项目。企业提供了进行项目的业务操作基础设施（设施、设备等）、员工、政策和程序，以及所必需的资源，无论是为了自己的利益还是为了与其建立了合同或协议的客户。企业管理团队关注的是企业的持续增长，其在行业中的声誉，以及在竞争激烈的市场中项目或软件产品的盈利能力。业务管理团队主要关

表 7-2　涉众需求的优先权

优先权	涉众需求
必要的	• 企业管理 • 顾客
很重要的	• 项目管理 • 最终用户
重要的	• 软件实现 • 软件测试和评估
显著的	• 开发后的过程 • 分配 • 训练 • 产品维护

注该项目的成功，并监控其完成既定的目标和任务的进展情况。它准许项目管理团队用权力和范围指导软件开发团队接近实现最能服务于企业与客户或最终用户的项目指令。

客户代表与软件开发企业签订合同事宜的实体企业。在某些情况下，客户可以是同一业务实体中的一个软件开发项目组织。在这种情况下，一种被称为协议的合同在客户和软件开发组织之间建立，因为客户组织依赖于软件产品来提高业务流程效率，降低运营成本或提高盈利。客户的需求和期望必须被优先理解，先于合同或协议中术语和条件的构建，正是在它们的基础之上软件产品被开发、存档并交付使用。合同或协议应产生成本、进度及在交付时软件产品必须表现的业务和性能特征。

项目管理是被授权计划、监督并指导软件产品的工程设计和开发及相关开发后的过程的项目监督团队。项目管理团队是与业务管理、客户和软件工程及开发组织的主要连接接口。其重点是项目目标和任务的完成情况，项目状态报告，与客户的互动向满足合同或协议的方向迈进，以及软件产品对客户、最终用户和其他涉众交付和接受程度。项目管理团队涉及项目规划和控制、配置管理和变更控制、成本跟踪和统计、风险管理团队和与合同及法律机构的接口。

121 ~ 126

最终用户代表将操作该软件产品的客户的员工或个人。最终用户所关心的是用户界面、应用交互、数据存储和检索、数据分析和报告生成。使用简单、无差错的数据处理和直观的交互是最终用户主要关心的。控制涉及硬件组件更大的系统的实时系统，关心的是当系统运行遇到不利条件时，系统的操作、故障模式、系统性能的缓慢退化、预警机制和纠正措施。

软件实现是负责将物理架构的结构单元规范转化成软件单元的设计编码并测试这些代码段，及进行软件组件的集成和测试的组织。它为 SWE-IPT 提供了代表，以洞察软件实现问题和不断发展的架构固有的问题。

软件测试与评估代表负责建立软件产品和开发后的过程中的测试计划和程序的组织。它为 SWE-IPT 提供了代表，以提供对不断发展的架构固有的测试问题和担忧的洞察。这也保证了软件需求适当地确定，以提供一个完整的、一致的和负担得起的测试策略；进行软件模拟运行和验收测试；执行软件质量保证检查，以确保遵守软件开发计划、程序和指导。

开发后期维护代表负责定义、设计、实现和检测开发后的过程的组织。这些过程的资格检测是测试和评估练习，确保该过程以足够有效的方法促进预期开发后期活动。这些组织为 SWE-IPT 提供了代表，以提供对不断发展的软件产品架构固有开发后期问题和担忧的洞察。该组织涉及精通软件分配、软件培训和软件支持方面问题的工作人员。

各涉众有不同的，往往相互竞争的利益、需求和期望，这些必须由软件工程工作来解决。这种多元化的利益造成设计挑战，可能使执行权衡分析来解决成为必要。涉众的需求和期望必须转化成软件产品和开发后的过程的一系列需求。这些需求作为一种或多种规范被记录，据此软件架构和测试程序将被开发。该要求规范不应划定基线（置于配置控制之下），直到需求不再变化并确定在项目成本和进度目标之内可以实现。

7.2　第 2 步：需求分析与规约

将涉众的需求转化成一个可行的、有效的软件解决方案，它包括对由客户提出的问题空间的探索和评估并分析潜在的设计方法。只有当设计方案充分成熟，技术风险消除或尽量减少，需求才应确定。这就要求软件工程实践应用来导出一个设计方案，该方案平衡了各种涉众需求与成本和进度的约束。一般的方法来导出软件解决方案涉及以下三个方面：

127

1）将涉众的需求和期望转化成软件架构。

2）平衡的、可实现软件需求的一套规约。

3）将解决方案合并纳入工作包、以及技术和项目规划和进度表。

涉众的需求和期望必须进行评估、协调和解决冲突，以确保涉众的期望在项目进入更详细的工程活动之前正确建立。抽象的解决方案范围由想法、考虑和通过候选方案限制问题解决的约束的方式产生。有了问题的界限就允许了解决最重要的涉众需求和技术挑战的产品概念的精心构想。最初的产品概念应该解决该产品在一个更大的业务或运营背景下怎样工作。应检查顶层或主要功能和性能的挑战，以获得对为实现产品的解决方案所需工作的程度的欣赏。最初的物理解决方案应该准备确定关键的软件产品的结构部件，并试图降低产品复杂性。

最初的产品概念应对业务情况、项目目标、技术可行性和产品维护的概念进行评估。某些技术解决方案可能会产生非常有说服力和令用户满意的产品。然而，软件实现和测试工作中技术上的挑战可能会增加产品的培训或支持成本。产品技术解决方案应尽可能不复杂以减少实现、测试和维护成本。在这关键时刻，产品概念应该由开发后的过程定义产生的整个生命周期成本进行评估。这些过程概念化并产生初步过程定义来支持这一评估。最终，最初的软件架构应该就产品和过程要求、初始功能和物理架构而被确定。完整的软件架构由产品和开发后的过程架构的组合代表。

涉及对涉众需求初步探索的挑战是：

1）平衡和化解冲突的涉众的需求。

2）维护项目的范围，以确保项目的成本和进度目标可以实现，其中包括以下内容：

128

- 与实现完整软件需求相关的成本。
- 与基于分类的操作线程、控制机制和外部系统及应用程序接口的测试软件产品相关的成本。
- 与定义和建立开发后的过程相关的成本。
- 软件开发时间表和必要的任务从属物来实现软件开发的每一个阶段。

3）确保拥有经验丰富的软件人才，且他们具有设计、编码和测试软件产品所需技术技能。

7.2.1　平衡和化解涉众需求的冲突

解决涉众的需求冲突，以便软件需求可以被确定，并了解了每个需求是怎样影响软件架构、项目成本和进度目标、产品的复杂性和开发后的过程的。与每个需求相关的成本和风险必须进行评估，来为产生可实现的软件需求提供基础。涉众的需求必须基于项目目标和产品可行性的潜在成本和风险的合作协议进行优先级划分。这些排列了优先级的需求，包括估计的成本、进度和风险信息，为解决涉众之间的需求冲突，并获得涉众对推荐的折中方案的认可提供了基础，这个折中方案形成了建立软件规范的基础。

对排列了优先级的涉众需求的进一步分析可能涉及功能分析与分配以及设计综合，以更好地理解软件产品的结果、开发后的过程，以及项目成本和进度目标。权衡分析和风险评估应进行在设计候选方案和允许设计决定之间区分，以产生更精确的信息。进一步的分析将提供支持数据和基本原理，用于协助涉众体会到他们含糊的或过度的需求的影响。

7.2.2　维护项目的范围

软件开发工作的范围必须保持在既有项目成本和进度的界限内。必须认识到，若干因素会影响项目的范围和规划，如：

- 涉众需求或软件需求的模棱两可导致影响技术工作范围的误解。当工作人员认定他们了解涉众的需求，并且没有带着消除可能包含多种可能的意思或解释的声明的意图去仔细检查每一个需要或要求，就会产生歧义。软件专业人士需要学科专家的援助，他们熟悉软件产品被期望来发挥作用的行业或业务环境。这些专业知识将减轻与解释涉众需求相关的模棱两可。 129
- 随着软件开发工作的进行，涉众的需求和期望发生了变更。这表示了与涉众需求的理解和明确相关的演变，同时它也影响软件产品。产品需求的频繁变更会削弱项目团队稳定软件规范和推进软件实现的进程。
- 对新加的功能和特征或对旧软件功能和特性修改的纳入相关工作范围的低估。每一个变化必须全面检查，以确定将该变更包含进产品架构、相关文档和技术计划带来的影响。

项目范围的每个变更都会对既定的软件开发工作产生不同程度的影响。软件工程的一个目的就是，以在项目成本和进度界限内可以实现和测试的方式，制定产品结构。初步设计审查（PDR）或关键设计审查（CDR）之后，项目时间表后期的变更将需要修改软件规范、软件架构和测试计划与程序。项目范围的变更必然会导致整个项目工作需要修改软件规范、架构设计文档和技术方案。否则，该项目在工作包方面超越预期工作将受到挑战。混合范围内变更的行为也影响该项目，通过阻碍技术进程以确定调整软件架构、重新编程和重新分配资源，并更新项目和技术计划来反映引入的变更所需的工作。

1. 与实现完整的软件需求相关的成本

每个软件需求都涉及成本和进度的责任，这影响着既定的软件开发工作。必须理解渴望实现的功能和性能水平，以便恰当地制定满足需求所需的工作计划。WBS 必须为软件需求追溯到项目和技术方案提供基础。范围变更提出的越晚、变更带来的影响也就越大。这是由于重新评估架构影响、重新设计、编码，以及测试现有的软件单元，重新集成和测试软件组件，将变更容纳到技术方案及文档中的需要。这应该被视为一种改进一项已经生成的工作产品以适应新的条件所需的返工。

根据经验，软件需求应反映软件产品在行业或业务流程中的作用。这些需求应该在数量上是最少的，但充分、清晰地被表达，确保当受到不同程度的业务压力时，测试策略恰当地演示出产品性能的有效性和效率。业务概念必须被转化为技术术语来指导一个完整的、有效的、不复杂的产品架构的建立。关于一个业务模型怎样被用来帮助建立一个坚实的需求基线 130
的内容将在第 8 章提供进一步指导。

测试是依据软件需求规约执行的，意在证明该产品符合已经确定的需求。如果规约没有准确地掌握涉众的需求和期望，那么交付的软件产品可能不能为所有的涉众提供有价值的建议。为确保软件产品能满足涉众，测试策略、计划和程序应确保在各种场景中和有压力的多用户业务条件下，与业务概念一致。

2. 软件产品测试相关的成本

软件测试和评估工作的工作量的确定必须适应项目范围和资源的限度。测试出现在软件

实现的各个阶段，必须强调指定的性能度量的成绩。功能需求确定了什么是软件必须做的，而性能度量确定某功能必须被完成到怎样好。一些需求可能被指定为在没有特殊的测试设施或设备时不能验证。其他的可能在交付之前达标成本过高。如果某个需求妨碍了明确的或成本效益的测试方法的构建，那么这个需求应该在测试计划中被突出，以改善这种情况。

这方面的一个例子就是一个基于互联网的游戏，允许数千实时用户在数字世界里互动。用预期数量的用户进行性能测试可能是具有挑战性的，且成本很高。因此，在极端情况下评估软件产品性能的方法，利用先进的负载测试工具和技术，肯定能够推断和预测性能度量。

3. 定义和建立开发后的过程相关的成本

集成产品和过程开发（IPPD）的理念强调同时完成建立开发后期进程相关的工作的必要。除非开发后的过程已经被定义、设计、实现和测试，否则软件产品不应实施。这些过程为软件产品分配、培训用户，以及提供客户和软件支持提供了必要的基础设施。软件开发项目应该围绕资源和管理监督，这也是那些基本的软件开发后的过程的必要条件。

4. 软件开发时间表和任务依赖

项目和技术计划、预算和进度表的精确性涉及管理过程，管理过程对项目范围和产品架
131 构设计变更进行反应。最初的计划永远不会考虑到开发工作的真正范围，因为对涉众需求的理解可能是有限的。软件工程的目的是研究涉众的需求和期望，从而可能形成一个技术解决方案。随着软件产品架构的开发，对技术工作有了更深刻的洞察，并且计划和进度表的精确度也提高了。项目和技术计划必须不断修订和更新，以反映从软件工程工作产生的改进的信息，而不是受最初的规划设想的驱动。在软件工程工作中做出的设计决策将要求项目团队不断重新评估计划、工作包和风险，来重新分配预算，重新调整任务进度表及依赖关系。

7.2.3　有经验的软件人员的参与

拥有能干的、有经验的并受过训练的软件专业人员，是执行技术方案的关键。任务持续时间和成本会随技术人员的专业知识的水平而改变。软件实现、测试和评估，以及后期开发过程组织的代表必须经过适当的培训来执行技术方案中确定的任务。此外，这些人员必须接受有关不断发展的架构设计的教育，因为这影响了他们的技术职责。工作包必须建立和修订，考虑到软件专业人员的参与、他们的综合技能，以及他们对软件产品性质、开发方法、程序和自动工具的使用的熟悉程度。

技术和项目计划必须加以修订，以准确地反映演变的软件需求和产品架构。当涉众需求和期望被转化成软件需求、接口和开发后的过程规范时，这些文档和支持分析为精炼执行每个开发任务所需的范围和工作提供了基础。项目计划和管理控制必须一致，以反映开发项目当前最准确的状态。随着软件工程工作的进展，引导设计决策的分析，对每项任务的范围、资源需求、技能，以及执行每个任务必要的工作水平提供了一个更实际的估计。

7.3　第 3 步：任务定义与安排

每个需求都必须被 SWE-IPT 评估，来确立组织角色、责任和任务定义。软件工程实践应该用于获得通过软件产品来满足的需求功能和性能特征的完整的理解。软件实现和测试组织必须利用这个理解来重构他们的任务范围。组织任务应该进行评估，以识别出任务性能相
132 关的潜在的风险和不确定性。应该为每一个风险降低方法确定应急计划，并且必须确定允许它们激活的条件。必须均衡任务之间的依赖，以便建立任务初始准则并提供整体工作流和进

度表的一个视图。

7.4　第 4 步：资源的确定、估算和分配

执行每个任务所需的资源必须从这些方面来确认，包括人力、设施、设备、办公用品以及计算机自动化软件工程（CASE）工具。资源可以包括产生项目成本的任何项目，包括旅游费用、复制和软件相关的技能培训。与资源相关的成本应该分配到参与执行任务的技术组织。低级任务应该联合起来建立高级任务成本估算。这种活动导致了对组织的与整体技术的工作包的预算评估的鉴定。为总结执行每个 WBS 元素的成本，可以将工作包进行组合。

最初的工作包成本估算必须与可用项目资源是一致的。任务范围和资源需求可能需要修订，来提供与工程预算约束一致的、一组完整的工作包。包含着已经确认的风险的工作包应该包含一个应急储备预算——基于风险的大小、发生的可能性，以及执行行动的应急计划所需的资源。

7.5　第 5 步：建立组织工作包

WBS 是一个核心的项目管理工具，与项目计划、预算和进度表紧紧相连。每个技术组织都应该保持自己的 WBS 版本，每个版本确认组织任务的描述、资源和结果。技术组织必须将他们对任务执行的贡献转变为一组有组织的工作包。这些有组织的工作包形成了组织和技术规划的基础。

7.6　第 6 步：技术规划

每个技术组织利用其 WBS 来产生或修订组织的计划和进度表。这些计划应该确定与每个工作包相关的组织角色和任务。任务描述、依赖、持续（开始和停止时间）、资源需求、期望结果、风险，以及应急计划都应该确定。SWE-IPT 负责确保组织计划跟既定的项目目 133
标、预算和阶段是一致的。必须通过结合组织计划、消除重复工作、准备集成工作包定义来建立一个集成的技术方案（ITP）和进度表（ITS）。

必须编写或修订软件工程计划（SWEP），以便为软件开发迫在眉睫的阶段获得详细任务。通过确定为正面对的开发阶段开发所渴望的结果的方法，SWEP 确定了 SWE-IPT 的焦点。它应确定任何已知的将被监控的风险和将被发起的行动应急方案，它将计划的行动改成另一个备选设计解决方案。为执行确定设计挑战的业务学习的计划应该被表达，竞争性的备选方案应该被描述，选择的标准应该建立。

主要的软件工程活动包括需求分析、功能分析和应用设计综合。它们是由软件分析、控制、验证和确认活动支持。软件分析涉及风险评估和权衡研究的进行，并在需要时随时应用。控制、验证和确认活动在软件开发的各个阶段都不同，因为工作的重点从涉众的需求和期望到软件架构、实现和验收测试不断发展。SWEP、ITP、ITS 和组织计划必须更新，以反映随着在每个阶段的进展，软件开发工作的演变。

7.7　第 7 步：项目规划

拥有首席软件工程师支持的项目管理团队，应该利用技术计划来建立或修订项目计划和进度表。项目管理活动的工作包，如项目控制、成本跟踪、配置和数据管理，应该跟技术工

作包描述集成。这些集成的项目工作包被用于建立 IMP 和 IMS，项目级 WBS 和预算。

一般的建立项目计划的流程旨在确保技术和项目计划准确地反映涉众的需求、特定的软件需求，并致力于项目目标的实现。这种方法是由涉众需求分析活动推进的，分析活动包括探究问题域、评估软件设计解决方案和备选方案。一旦软件解决方案被承认，必须修订项目、技术和组织计划，以提供要执行工作的准确、一致的表现。规划序列强调了进行充分的软件工程分析，来产生指导技术和项目规划工作的软件需求规范的重要性。这是为了确保软件开发工作是由涉众的需求和软件需求驱动的，而不是项目开始形成的有错误或不准确的项目计划。

7.8　探索涉众的需求

只有当涉众的需求和期望都正确地转化为一套可实现的软件需求时，才能制定出详细的技术方案。引出、协调和按优先顺序排列涉众的需求，是将工作计划与项目目标相结合的重要步骤。然而，软件需求确定之前，必须进行一定量的设计概念化和探究，以确保产品可开发，且在项目资源约束内能够交付。软件需求基线描绘了技术、项目管理，以及有关产品特点和业务有效性的涉众代表之间具有约束力的协议（有时是合同）。为确保项目团队能成功交付软件产品，实现项目成本和进度目标，必须花费一定量的努力来增强架构概念。早期和持续软件架构检查，特别是在技术上具有挑战性的或危险的部分，提供了技术和项目计划一定能满足的保证。

建立软件产品架构涉及执行软件工程实践的几次迭代。在概念上，软件开发的首先的三个阶段的每一个都涉及执行这些实践的大部分至少一次，以产生需求规范及功能和物理架构。图 7-2 显示了软件工程实践怎样应用于为开发的每个阶段产生所期望的结果。然而，在每个阶段这些实践可以重复，在必要时修改、提炼或完善软件产品架构。

软件开发首先的三个阶段的每个都是为了使软件架构定义朝着足够引导软件实现的特异性的状态发展。不幸的是，这些阶段的标题与软件工程实践的标题相似。这种相似性引起了混乱。软件开发从业者错误地假定只有确定的软件工程实践是在这些开发早期阶段执行的。然而，在为了消除风险或创建一个设计解决方案的开发的每个阶段，根据需要应用软件工程实践是至关重要的。

从历史上看，初步设计阶段产生了要求在整个功能层次的需求分配。功能组件和单元被不适当地认为是初步设计配置的元素。作为设计介质，类似于功能标记，功能和物理结构之间的区别由于软件没有被理解。软件语言包括类似于数学或逻辑函数的语句。早期的编程语言包括程序、函数和子程序，作为它们基本的结构元素。因此，软件从业者很容易假定架构定义在功能充分分解时就已经完成了。然而，必须知道，每一个产品都有物理配置。功能分析与分配实践使软件需求被转化成软件产品肯定能执行的基本数据处理活动。但是，这并不能断定该软件架构的定义。必须综合物理配置，以提供一个条理分明、不复杂的解决方案。

当它的影响范围被问题 / 解决方案空间的划分牵涉到，每个软件工程实践都肯定会被利用。这种软件工程实践的迭代肯定也总要重新回到需求分析活动，来修改或改善软件产品需求。例如（参见图 7-2），在开发过程中的详细架构定义阶段，需求分析实践出现了两次。第一次出现确定了结构单元的规范，第二次出现确定了结构组件的规范。

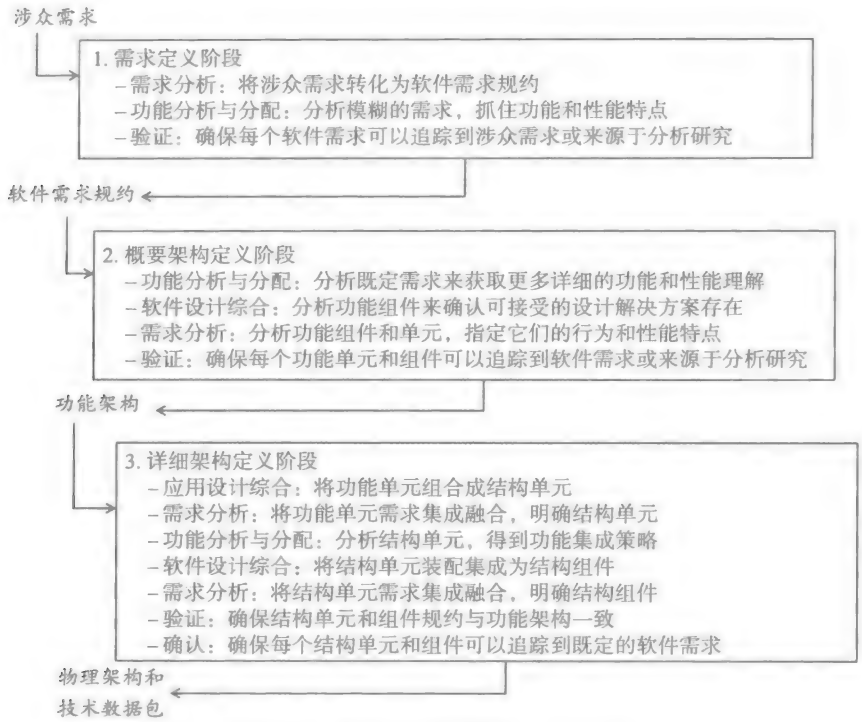


图 7-2　软件工程实践概念上的应用

在软件需求定义阶段，可能有些涉众需求需要建模或创建原型，以确保涉众的需求被理解，探究设计挑战，并尽量减少开发工作的风险。这些分析工具应该用来确保软件要求被清晰地说明。这项工作必须被认为是为了澄清涉众需求目的，对设计范围的“涉猎”。术语涉猎用来强调它表示背离软件工程实践正常的或计划的应用。模型或原型开发应该通过为澄清软件要求的目的建立模型或原型的软件工程技术的快速应用来实现。模型必须仍表现出某些提高的需求规范准确性的功能和性能特性。原型将涉及软件实现的工作，以开发一个模拟的原型设计。这些模型和原型必须被视为促进对涉众的需求增强理解的工具。

134 / 138

第 8 章

Software Engineering: Architecture-Driven Software Development

软件需求分析实践

软件需求分析就是软件工程实践中，在软件架构的顶层，将涉众的需求和期望转换成可行的一套软件需求。随着软件产品架构被系统地揭示，这些任务应该被选择性地应用于为每个架构元素指定需求。这组任务适用于软件产品（应用程序和计算环境），以及开发后期软件维护过程。软件需求分析包括了一套分析和评估，审查涉众的需要和软件需求，以理解开发工作范围内的每个需求的含义。

在需求确定了基线并处于配置控制之下之前，必须理解与设计解决方案相关的复杂性。此外，需求分析必须评估，在项目成本和进度的约束下，满足所有软件需求的可行性。这只能通过探索每个需求的解决方案空间，并关联各种性能级别的项目变量敏感性来完成。这可能使得进行探索性架构研究（功能分析和应用设计合成）和权衡分析成为必要，这样来完成一套能在项目成本和进度目标内成功实现的平衡的需求。

本章讨论，在给定了限制的应用运行环境情况下，与软件需求分析相关的主要任务。这涉及业务和维护过程中的研究，以确定需要用来满足软件产品涉众的功能和性能特点。这些软件需求分析任务的执行产生了软件产品的规范（计算环境、软件应用、应用程序接口）和软件维护流程的规范。必须通俗地来调整这些规范，直到产品设计确定足够明确，并且涉众同意置于配置控制之下。软件需求基线和接口规范形成了软件测试和评估工作基础。图 8-1 描绘了有助于软件需求分析的任务。

这些任务已经被分类，来强调每个任务确定的研究领域。这些类别包括项目、业务、产品、开发后的过程分析和项目评估。这些分析任务旨在检查软件产品，在其项目、操作和维护环境的情境中了解它。项目评估任务按最初的需求执行，来了解如果需求被批准、确定基线，并置于项目配置控制之下时，实现项目目标的可行性。

8.1 项目分析任务

项目分析任务包括收集有关待开发产品的信息，来确定产品和项目成功的标准。软件需求规约必须保持在项目能够成功执行的范围内来完成。

8.1.1 分析项目目的和目标

每个软件开发项目都有一系列的目的和目标，指导着项目团队的工作。由于缺乏对最终产品的清晰认识，初始的项目目的和目标本质上是抽象且浮夸的。随着开发工作的进展，开发团队对涉众的行业或业务挑战有了更好的理解、评估和解读。这种理解必须用来增强项目的目的和目标，以及恰当地说明软件产品需求。

项目目标必须以将项目团队的注意力集中到项目结果的形式呈现。目标会形成组织的结构和计划，所以每个任务都会以某种方式促进目标完成。目标必须是可度量的，并且描述的是可实现的成果、行动结果以及相关的项目里程碑。软件开发目标应努力达成项目和企业的

期望。目标根据通过执行开发工作得到的益处，达到项目的目的。成功因素应该包括那些项目想要达成目的或目标必须执行的活动要素。

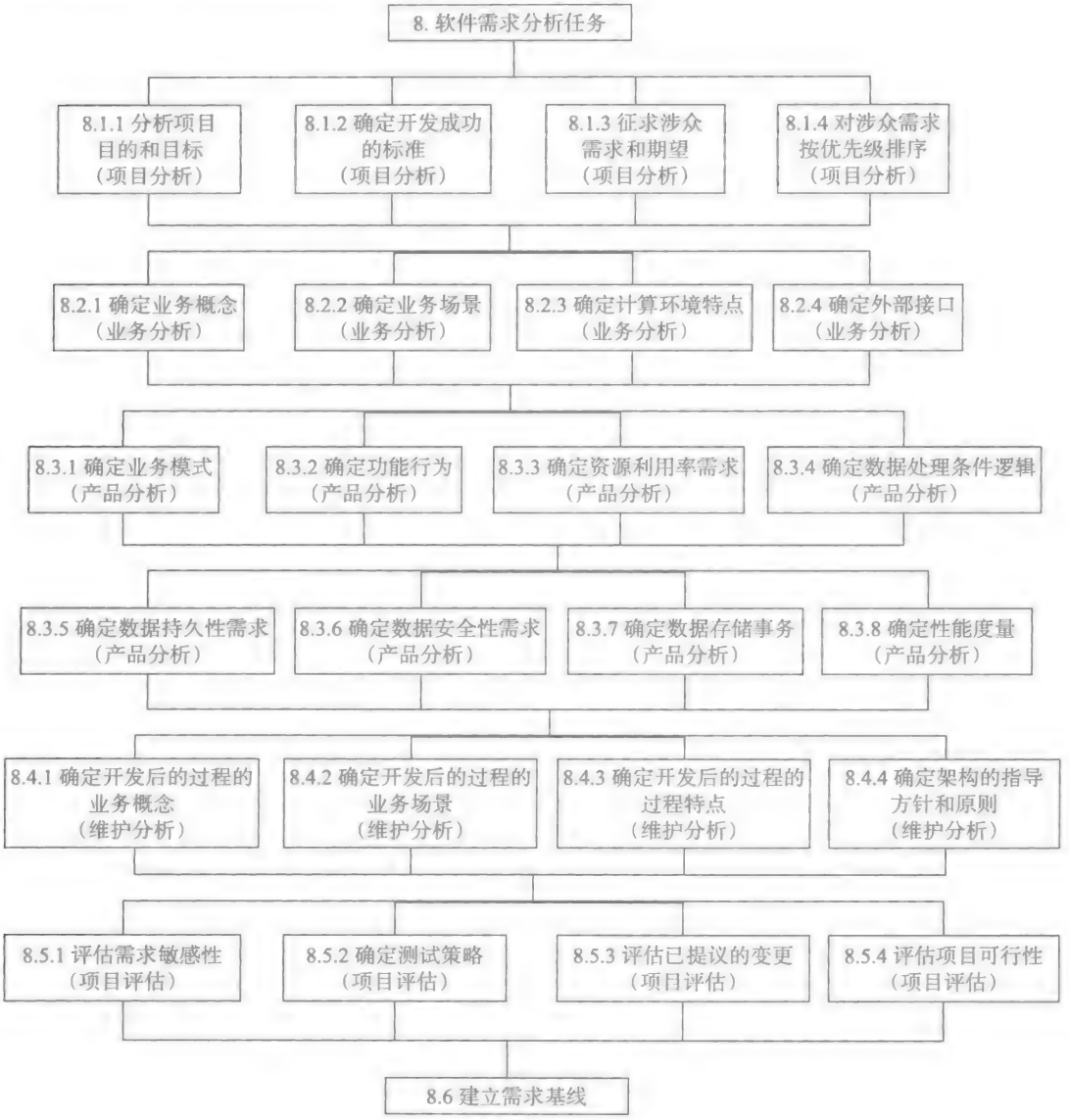


图 8-1　软件需求分析任务

8.1.2　确定开发成功标准

必须为每个成功因素建立软件开发成功标准。这些标准必须确定一个成功因素的满意度如何被衡量或认识。成功标准建立了指标，通过这些指标，项目会在涉众的眼里被判定为已经成功。以项目为导向的成功标准应该针对那些关键的项目管理事件，它们证明了该项目的管理和控制得当。以结果为导向的成功标准则针对作为执行项目的结论实现的结果。成功标准必须被监控，以便能够判断一个项目是否兑现了涉众预期的利益。

8.1.3　征求涉众需求和期望

对开发在交付的时候可以被接受的解决方案来说，了解客户的需求和期望是核心。历史是由交付时间晚、超出计划成本约束，或者在最后交付时被客户拒绝等有瑕疵的软件开发项目构成的。需求蠕变——需求的不断变化和增加——是项目面临的一大困境。软件开发经理并不能阻止客户在整个开发过程中试图不断更改或添加需求。必须认识到，每一个需求意味着开发工作的一项开销，其中包括了测试成本的增加和对交付日程的影响。

141
~
142

从根本上说，客户没有需求，他们有的是需要和期望。当面对一项新需求相关的成本时，客户往往会改变他们既定的需求，或完全放弃该需求。软件开发经理面临的最大挑战是建立一个需求基线，并且不允许它改变。随着软件工程过程的执行，将出现衍生需求，它们影响开发工作在预算内、按时交付解决方案。允许需求变更或添加到工作中，只会使这种情况更糟糕。如果一个需求要更改，那么它应该是通过减少或消除程序风险，以一种简化开发工作的方式来改进。

必须掌握客户的需要和期望，以作为要为计算环境和软件应用指定的需求的来源。我们应该根据期望解决方案如何操作或支持客户的业务流程来掌握客户的需要和期望，并且尽最大努力将数量降低到最少。客户的期望不应该表示解决方案中的设计约束，而是应该确定访问和处理数据的方式。

软件开发项目通常是在企业组织内部建立的。企业代表了以既定的商业目标为目的的商业实体、政府或社会组织。所有企业都有政策和财政责任。没有按照预期执行的开发计划会影响企业的财政稳定性。当程序建立后，企业会确定成本和进度目标，来约束软件开发项目能够完成和交付什么。此外，政策会决定技术审查、质量计划、技术使用等的执行，所有这些都会影响开发工作和最终的解决方案。

必须捕获企业和项目的约束条件，因为它们会形成管理软件开发工作的基础。约束条件影响解决方案空间，并且会通过限制设计的选择余地来加重开发工作的负担。标准、计算机辅助软件工程（CASE）工具、网络，甚至计算机语言的使用都会影响开发工作，影响成本和进度。这些约束必须作为涉众的额外需求来捕获。这承认了企业和项目管理团队也是附加的客户，他们有自己的期望，是软件开发程序成功中的涉众。

外部约束是指与开发项目要解决的问题相关的法律、法规或行业政策。外部约束影响解决方案的空间，并且为了设计方案在政治上和社会上是可靠的，必须服从外部约束。在需求确定基线之前，识别出影响设计方案的所有可能的约束是很重要的。外部约束是一种不能偏离的需求，然而软件需求可以变更或在软件开发团队、企业和客户之间协商。

外部约束的一个例子是公法，它要求视频游戏行业根据基于流血、暴力和色情内容推荐的年龄限制，评定每个游戏的内容。娱乐软件分级委员会（ESRB）的政策旨在提供准确、客观的计算机和视频游戏信息，以便父母能够做出明智的购买决策。[⊖]

143

由于游戏行业是个自律组织，ESRB 负责其评估系统的执行。经过 ESRB 评级游戏的每一个发行商，当为 ESRB 评估提交该游戏时，从法律上讲，必然要公开所有相关内容。游戏公开发布之后，ESRB 测试者会审查最终产品，确保所有相关的内容得到了充分披露。如果发现有影响评估工作或内容描述的资料没有公布的事件，ESRB 有权强制其整改和实施广泛的制裁，包括罚款。整改行动包括撤下广告，直到评估信息被改正，用正确的评估信息重新

⊖ 详见娱乐软件分级委员会，*http://www.esrb.org/*.

包装或召回产品。

这种约束要求游戏软件开发商引进一个任务，即在视频游戏发布到市场之前通过 ESRB 评估。要求软件包装必须清楚地显示评级，并跟 ESRB 准则是一致的。这可能会影响该产品的发布日期，且必须考虑到软件开发项目计划和进度表。此外，就目标受众和游戏包含的相关内容而言，评级系统会影响游戏的设计。

8.1.4　对涉众需求按优先级排序

项目团队必须精简、排列和总结涉众需求的列表，汇成一个易于管理的集合，然后用于引导软件需求规约。涉众的需要和期望必须转化成产品（计算环境和软件应用程序）的软件需求和开发后的过程。单个涉众的需要可能涉及项目要负责实现的几个需求。需要确定满足涉众所需要的工作水平和必要资源，项目和技术风险也必须列入成本预算。这样项目的预算就可以根据优先需要进行分配。这建立了软件开发工作的初步范围，并提供项目和技术规划的基础。

8.2　业务分析任务

业务分析任务包括收集软件产品旨在促进的业务或操作流程相关的详细信息。业务或操作流程的特点应追溯到涉众的需要。应该根据设施、系统、设备、数据网络等来确定操作环境，软件产品也将会在这里执行。业务分析应该从软件产品的角度来描述，并确定其角色、功能，以及与外部系统、人员、资源提供者（比如电源、数据源）的互动。 144

在架构定义阶段，业务模型阐述为在业务环境中确定软件产品行为。扩展业务模型将在第 11 章讲述。

8.2.1　确定业务概念

软件产品的业务概念必须文档化，以描述软件产品的特征，而这些软件产品是从产品使用者的角度提出的。它用来将软件特征传达给所有涉众。业务描述概念通常涉及以下内容：

1）软件产品目的和目标的声明。

2）表述产品所提供服务的任务说明。

3）影响产品的战略、政策和约束。

4）组织、活动以及参与者和涉众之间的互动。

5）与产品开发和维护相关的明确责任声明。

6）产品的分配、培训和维护的过程识别。

7）重大事件决策的规定和权威。

在软件需求定义阶段，应该对操作或业务流程建模，以了解计算环境中的元素、相关的应用程序、数据库和用户间的业务活动流、控制流和数据流。业务概念应该为软件应用要支持的每种业务都确定操作线程。

在架构定义过程中，业务概念应该确定用户与产品的安装、操作、分配、培训和支持之间的互动。

8.2.2　确定业务场景

应该扩大业务概念，以根据业务场景确定预期的软件应用用途。对于每个业务场景，应

该明确所期望的与其他系统、产品或用户的预期的交互。每个事务的业务规则都应该被捕获，并表示为控制决策逻辑。此外，每个业务场景应确定会阻止事务完全成功的可能场景，确定事务回滚程序的需要。（注：这代表一个衍生的需求，在最初的需求集合中可能没有得到处理。）

145 鉴于大多数计算机系统有某种形式的自动诊断，应该确定故障排除诊断线程。这些故障排除线程应确定，当抑制特定类型的事务，或者当系统完全退化的故障发生时，将要采取的行动。

必须为实现业务目标的软件解决方案（软件产品和计算环境的组合）的所有性能确定有效性的度量。根据解决方案按业务场景表述的那样执行自己任务的能力，有效性的度量确定了解决方案预期的有效性。业务有效性是考虑到整体的业务环境，软件解决方案成功完成任务的综合能力。[⊖]

有效性的度量应该指定为最低限度的可接受水平（临界值）和预期目的（目标）。这提供了所需性能的范围，基于这个范围设计计算环境和应用程序。图 8-2 给出了有效性度量的元素。对于汽车来说，有效性度量可以是："汽车在高速公路行驶时，每加仑汽油应该能跑 36 英里。"这句话看似简单，但有几个影响实现这一设计目标的设计因素。重量、马力、燃油效率和阻力，这些是为达到预期的有效性度量不得不合理利用的一些设计考虑因素。

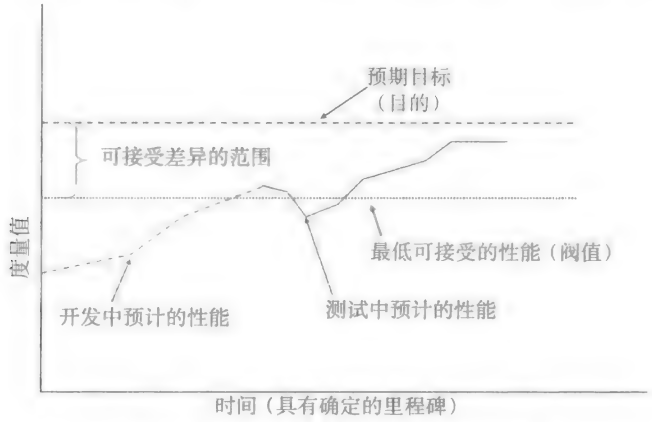


图 8-2　明确定义的有效性度量的元素

8.2.3　确定计算环境特征

146 必须确定计算环境，明确软件产品在联网、协作或者多用户操作环境中运转的能力范围。计算环境特征应该确定计算主机、服务器、工作站、数据存储设备、绘图仪、操作系统以及其他应用软件，比如数据库管理系统。这些信息是计算开发的定义基础所必需的，且为了支持软件测试，必须实例化。

确定计算范围是必要的。计算范围可能包括本地、广域、无线和电信网络。建立计算范围是用来了解软件产品需要怎样与计算环境各种元素和其他外部系统进行互动。对嵌入式软件产品的情况，这个范围可能是其运行所在的系统。然而，如果系统是一个更大的"系统的系统"的一部分，那么计算范围可以扩展到超出系统边界，直到表明所需外部接口的其他系统。

⊖ （2001）. *Systems Engineering Fundamentals*. Defense Acquisition University Press.

8.2.4　确定外部接口

必须确定与外部系统的接口，需要指定与每个接口相关的需求。外部系统是那些没有作为该程序的一部分被开发的系统，但它们需要与正在开发的软件产品交换数据。接口规约应该不仅是针对数据的传输物理形式，也应详述信息的格式、数据类型、度量单元和与每个数据类型相关的数据精度。

应该分析业务模型，以便为业务、支持和诊断行为分别指定各个人机交互。此外，人机交互接口需求应解决应用程序如何发现和处理不恰当或不正确的手工数据输入。应该确认手动交互的方式和交互设备，例如键盘、鼠标、触摸屏以及读卡器。人机交互界面需求也应处理显示给用户的数据，和由软件产品来生成的输出报告的类型。

8.3　产品分析任务

产品分析任务包括鉴于其在业务概念中的作用，详述软件产品需求。业务概念和场景应该进行评估，以确定与该软件产品必须表现出的数据处理操作相关联的功能和性能特点。关于需要由业务模型来表达的元素细节在任务 8.3.2 中进行了讨论，确定功能的行为（见图 8-1）。

在架构定义阶段，产品需求会分解并分配给功能和结构单元和组件。这些任务应该有选择地执行为软件产品架构的每个元素指定需求。 147

8.3.1　确定业务模式

大多数应用程序的设计必须解决正常、退化、维护和训练模式，在不同的模式中进行操作。应该评估业务模型，以确定哪个业务模式决定了将应用从业务的一种模式到另一种模式的特定条件。例如，如果一个 ATM 银行自动取款机用完了现金分配，就不能进行提款交易，但它仍然可以处理其他的业务需求。然而，当在银行自动取款机正在进行维修更换磨损部件或在补充现金、收据和其他消耗品时，银行自动取款机将不能提供服务，且没有交易能够进行。

独立的业务模型可以为每个备选业务模式，或者该业务模型可为每个业务模式进行开发。最初分离的业务、培训和支持模型有时候更加有效，这使每个模型都有一个明确的焦点。一旦完成并验证，如果需要，这些模型可以被集成或组合到单个模型中。

8.3.2　确定功能行为

必须定义软件产品的功能行为，并使之与有效性的度量和先前定义的业务线程关联。业务行为应该描述组织要素如何通过执行活动或工作流程与软件产品相互作用。业务模型应该表达控制逻辑来执行业务规则、政策和步骤。信息和组织要素之间的交换应该表示为活动的输入和输出。

业务模型应该被开发来描绘与每个场景相关联的运营或业务工作流程。在经营模式开发阶段，支持运营过程中的系统应该作为一个参与者被确定。图 8-3 提供了一个业务行为模型的例子。产品的行为将在第 11 章进行进一步探讨。业务模型涉及以下元素：

1）组织元素代表了企业内部的组织或外部组织，如业务合作伙伴、供应商和用户。每个组织元素代表业务模型中的一个角色，并且用于对每个组织必须执行的活动进行分组。每

个组织元素通过执行以时间为序列的工作流程的业务活动来参与业务过程。

2）参与者代表一个特定组织角色、用户类型、系统、计算设备项或参与了业务过程的应用。参与者执行活动并与其他参与者交换信息。

3）业务活动代表了由一个参与者所执行的任务或工作。一个业务活动代表进行操作或业务过程中将输入转化为输出的一个步骤。业务活动在执行时可能需要可用的资源。业务活动与软件功能类似，但是从业务过程的角度表达，由参与者或组织元素执行。

4）流程控制机制的业务规则和控制机制确定处理中哪一步骤将在下面执行（例如，If（条件）Then（执行函数 X）Else（执行函数 Y））。控制机制可能代表一个重复以前执行的活动一次或多次的循环条件，或有条件的分支，其中每一分支包括适当的响应预期情况的活动。如果客户申请信贷，那么在初始信用评级搜索时，应用程序可能会被批准、拒绝或受到进一步的信用检查。每个结果是依赖于初始信用检查的结果，并改变相关联的后续活动的过程。

5）数据项是代表组织元素间的信息交换的业务活动的输入和输出，参与者发送或接收交换信息。数据流对于捕获非常重要，因为它可以代表以下数据流类型中的一种：

- 点对点——一项业务活动的输出数据项作为另一项业务活动的输入。
- 广播——一项业务活动的输出数据项作为其他更多活动的输入。
- 触发——一种启动接收业务活动的点对点数据项。接收业务活动不能开始执行，直到触发数据项被接收。
- 数据记录——一种发布到数据源或从数据源中检索的数据项，如数据库、文件柜或外部应用程序。数据记录代表有用的信息将进行保存，并支持检索业务运营。

6）期限——每个业务活动都消耗时间，当业务模型被分解时，可以生成业务时间线来表达业务过程执行需要多长时间。由于大多数自动化程序的目标是使这个过程更有效，成本更低，执行更迅速，所以当涉及软件产品时，理解现有流程执行时间和期望的性能上的改善是非常重要的。

7）资源——业务活动可能需要资源可用于要进行的业务活动。一个需要资源的不可用将导致活动被置于等待状态，直到该资源可用。资源可以是以下类型：

- 可消耗的——业务活动被执行时用完的资源。消耗的资源必须有一个可用的初始量，这可以是手头上在任一时刻的最大量（库存容量）和该活动所消耗的量。自动取款机拥有它可以提供的金钱的储备。每次取款交易都会减少被分配为消耗资源的可用钱数。
- 可重用的——由一个业务活动捕获并由该活动完成执行时释放的资源。收银机是一种可重用的资源类型，在同一时间只有一个销售合伙人可以使用登记簿进行交易。

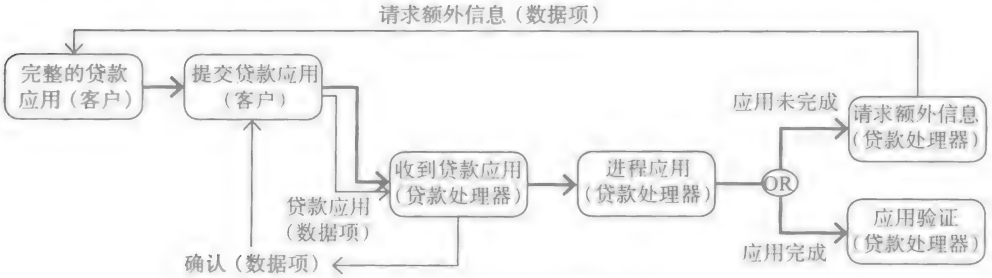


图 8-3　业务（行为）模型示例

8.3.3　确定资源利用率需求

业务模型应该进行分析，以了解资源可用性是如何影响业务和事务流程的。重点应放在计算环境资源，这涉及任何物理或虚拟组件有限的可用性。业务模型中，并行进行的活动可能会争夺可用资源。争夺资源会导致程序延迟，这可以通过增加资源的可用性得到缓解。当资源有限时，就可能需要调度机制，以确保高优先级程序在低优先级程序之前使用资源。

在业务或事务流程水平，资源使用分析在确定资源纳入计算环境方面是有用的。在分析的软件产品水平，资源的可用性可能约束数据处理流，并且可以确定修改的计算环境约束的需要，以并入额外的计算设备。

8.3.4　确定数据处理条件逻辑

通过评估业务模型来确定软件产品怎样解决过程控制机制。由软件产品展示的过程控制必须以这样的方式表达清楚，即明确确定进行查询的条件，并当标准都满足时执行后续行动。

148
~
150

8.3.5　确定数据持久性需求

应该对业务模型进行分析，以确定数据存储和事务处理需求。这些数据的持久化要求应该根据为数据库管理功能选择的硬件和软件，确定计算环境的特性。这些要求必须处理数据库管理系统（DBMS）的可靠性度量。对大多数软件应用程序和典型的 DBMS 能够事务回滚和备份性能来说，数据持久性和存储是一个重要的功能。

8.3.6　确定数据安全性需求

对于数据安全性的要求，必须规定：确保实施正确的数据访问和控制程序。数据安全性的要求必须进行评估，以确保商业和个人数据不能被破坏。业务规则中必须指明：控制谁可以在业务操作中从永久存储中访问、修改或者删除数据。对于每一级用户对数据访问控制的特权必须要指定，还有访问控制监视和强制执行的软件需求也要指定。

数据安全性的一个重要方面是为敏感的数据元素分配一个安全分类识别器。不同的数据安全分类等级必须被定义，且分配数据元素到每个等级的标准也应该指定。表 8-1 列出了一些典型的私人和政府部门使用的安全分类等级。

表 8-1　典型的私人和政府部门的安全分类等级

商务领域	政府领域
公开	不保密的
个人信息	保密的
敏感	机密的
机密	秘密
私人	顶级秘密
竞争敏感	

此外，对应用程序在数据存储或传输之前施加加密算法的需要必须明确提出。必须确定可以访问数据的全体人员的安全级别以及用户的类别或角色，必须确定用户类别、密码和数据安全管理责任。应用程序是如何执行这些职责，提供人员和组账户的管理，保护个人数据和监视数据存取，这需要被详细指出以确保正当的数据的安全性。

151

8.3.7　确定数据存储事务

应该分析业务模型，以确定每个数据存储和检索事务的需求。应确定用于执行每个事务的规则和将需要回滚事务的条件。所选的 DBMS 应该被检查，以查明事务死锁情况的原因。

在功能分析中，这些潜在的死锁情况应进一步分解，以确定该软件产品将需要如何监控每个事务来检测死锁和这些情况应如何通过取消和回滚事务或使用了避免死锁发生的其他预防措施来解决。

8.3.8　确定性能度量

在架构定义过程中，有效性度量（MOE）应该被分解成性能的多种度量（MOP），即确定软件解决方案期望多好地实现每个 MOE。性能度量要专注于评估业务性能特征的度量，例如，作为应在一段时间内执行的事务的数目、数据转换的精确度和资源利用率。

8.4　维护分析任务

维护分析任务主要详细说明产品销售、培训和支持的需求。随着软件产品架构的开发，复制、销售、培训和维护过程的需求必须重新审视，以确保开发后的过程已经充分被说明。

8.4.1　确定开发后的过程业务概念

软件开发后的过程必须详细说明，并最终设计、实现和测试。影响软件产品的主要开发后的过程是软件复制、销售、培训和维护。软件产品是独特的，因为它们可以电子地分布在因特网，或包装并分销给客户或零售网点。

维护概念应该解决一旦软件产品被部署或分销，最终用户、客户、计算环境以及软件产
152 品将如何被支持。软件支持包括软件修改：修复已报告的错误和通过额外添加的新功能加强软件产品。系统硬件可能涉及预防性维护措施，这对防止硬件组件磨损和替换已经磨损的组件是必要的。软件产品的业务概念必须找出可能的系统硬件故障状况和维护情况，以确定应用程序应该如何反应。软件产品可能需要“状态感知”，使数据处理操作在系统进行维护行动时，可以被禁用或暂停。

维护概念必须被理解，以评定软件解决方案生命周期的成本。由于解决方案可能是某业务操作的一部分，所以计算环境和软件产品的支持成本肯定是可以量化的，并可能影响设计决策。

8.4.2　确定开发后的过程业务场景

业务概念应该被扩展，以便为每个开发后的过程确定预期业务场景的范围。产品销售的方案应该针对销售渠道、分销商、发放许可证、产品退货政策、保证书和软件补丁交付。产品培训的方案应该针对使用教程、训练演习和材料、用户手册、课堂教学材料以及第三方培训合作伙伴。产品维护的方案应该针对帮助台操作、问题报告、跟踪和解决，以及软件产品增强的发展。

有效性的度量必须为实现维护目标的开发后的过程解决方案（软件产品的销售、培训、维护）的整体性能被确认。有效性度量，根据执行其业务场景中表达的使命的能力，定义了解决方案期望的有效性。

8.4.3　确定开发后的过程特征

应分析业务模型和支持概念，以确定和详细说明对包装、销售和安装的要求。软件维护

涉及“支持、维护和运行系统的软件方面所需要的流程、程序、人员、原料和信息。”[⊖]这些需求可能涉及必须在功能分析和应用设计综合中处理的功能。此外，所选的用于实现每个流程的方法可能对产品发布的时间表以及产品和项目成本有影响。

包装可能通过外部资源就能完成，并可能包括媒体的复制（例如，光盘复制）、用户手册和安装说明再现、包装插图、产品包装和装箱发货。 153

销售可以通过内部资源运输、外部邮政服务或者互联网来完成。网络销售将需要一些工作来准备、测试和监控允许访问电子销售包的网站活动，以及提供电子支付的方法。

安装可能会在客户的业务地点进行，以正确配置计算环境来达到最佳的软件性能，或者可以通过组织的网络管理员来安装。可能使用商业软件安装包，并且将需要一些工作来准备和验证安装包在各种计算机环境和一系列的操作系统中能正常工作。

培训交付方法必须被确定并详细说明，以纳入与作为软件部署战略一部分的最终用户的培训相关的成本。培训方法的常见形式包括：1）实际动手操作的培训指导的课堂培训，培训员给用户展示软件是如何工作的以及如何执行常见任务；2）团体示范培训会，教练以现场示范方式给用户展示软件是如何工作的以及如何执行常见任务；3）基于计算机的培训（CBT），它允许最终用户完成互动课程，该课程通过执行常见任务的过程，以及互动软件就其表现和理解力进行测试的方式，来引导他们；4）基于书面的培训，最终用户完成关于怎样执行常见任务的练习册课程，该练习册常附有截图。无论选择哪种训练方法或组合，产品的培训需求都必须要详细说明。这包括确定用户文档和培训材料，这些材料是在软件部署准备审查阶段必须被完成并且可用的。

支持过程包括帮助台操作、问题报告、跟踪、分解、软件补丁部署和软件增强开发。执行这些流程中每一个的组织能力的建立必须进行规划和开发，以支持软件部署准备审查。这可能涉及软件开发（工程、实现和测试）设备和 CASE（计算机辅助系统工程）工具到维护组织的转变。需要转换项目的目录必须由软件需求说明书决定，为开发后的规划提供基础。

8.4.4　确定架构的指导方针和原则

需要建立架构的指导方针和原则，这将控制整个软件产品生命周期的软件架构的演变。这些指导方针和原则应针对软件产品的业务和质量目标，并应提供该产品的结构的管理理念和它适应计算环境的技术组件变化的能力。此外，这些指导方针和原则应针对在软件生命周期中不同参与者的观点（用户、设计师、测试人员、支持人员等）。该架构的指导方针和原则必须是可度量的，这样才能够评估功能和物理架构，以确保合规性。 154

8.5　项目评估任务

维护分析任务针对项目目标、计划和资源约束的软件需求的分析，以保证需求可以被满足。需求中固有的风险必须考虑，以提供一个可接受的项目成功率。

8.5.1　评估需求敏感性

软件需求必须进行评估，以确定该项目的生存能力受风险影响到什么程度。敏感性和风

⊖ 维护软件——密集型系统，2006 年 5 月，CMU/SEI-2006-TN-008, *http://www.sei.cmu.edu/reports/06tn008.pdf.*

险分析[⊖]与可能导致不利后果的因素或因素的组合有关。这些因素必须在项目框架中被作为风险或假设识别出来。该组软件需求可能需要被修改，以提高项目成功率。

8.5.2　确定软件测试策略

业务模型应进行评估，并且应该准备好初步的软件测试计划，以便根据软件需求和计算环境的详细说明来创建测试软件产品的策略和方法。软件测试计划应确定该软件的测试环境，并提供一个关系到软件需求和软件测试策略元素的跟踪矩阵。软件测试环境包括硬件配置、操作系统和该产品所满足的特定需求涉及的相关应用程序。

随着需求变得更好理解，这时开始测试计划是非常重要的，因为测试可以消耗高达 20% 的计划时间，30% 的开发成本。

基于软件开发者和用户的调查，全国每年软件测试的基础设施不足的成本估算范围为 222 亿～ 595 亿美元。超过一半的这些成本是由软件用户以错误避免和减缓活动的形式在承担。其余的成本由软件开发人员负担，反映了由不恰当的测试工具和方法消耗的额外测试资源。[⊜]

155

就像 NIST 报告所给出和图 8-4 所反映的，软件测试通常仅在测试阶段。然而，测试计划和测试用例及流程的建立在软件开发的整个早期阶段就出现了。软件测试必须被认为是意义重大的工作，其计划必须尽可能早开始。当软件需求在被分析、评估和正规化的时候，测试和合格化软件产品的挑战就必须被确定。软件规范的产品资格部分给出了分析、检查、论证和测试方法，这将用来基于测试结果，确认需求满足。软件测试的成本在决定整个软件开发成本中是一个重要部分。

	需求分析	概要设计	详细设计	编码和单元测试	集成和测试	系统测试
20 世纪 60 ～ 70 年代	10%			80%	10%	
20 世纪 80 年代	20%		60%		20%	
20 世纪 90 年代	40%	30%		30%		

图 8-4　软件开发各阶段的工作分配

8.5.3　评估已提议的变更

必须评估软件需求的变更，以确定相对于变更的必要性，该变更对项目架构的影响。如果一个被提出的变更被确定为对软件产品满足涉众需求是必要的，那么这个变更可能是必需的。然而，项目资源可能不能容纳将提出的变更纳入软件架构所需的额外工作。执行某变更影响的评估方法涉及以下 5 个步骤：

1）定义由确认出的工作包引起的变更的范围，这些工作包会受批准的变更影响。

2）确定拟建项目状态的主要区别，以确定其对项目实现里程碑和目标的关键路径的影响。

3）确定项目框架的能力，以通过适当的调整工作估计来适应所提出的变更。

⊖ 敏感性分析试图评估：如果某一假设没有或仅仅部分实现，这对实现项目目标的影响。风险分析评定某一假设没有或仅仅部分发生时的实际风险。

⊜ 国家标准和技术研究院（NIST），项目办公室战略计划，和经济分析小组，规划报告 02-3，基础设施不足软件测试的经济影响，2002 年 5 月。

4）假设变更被批准了，然后评估项目敏感性或成功率。

5）介绍对适当的变更控制委员会的调查结果和建议。

一旦需求基准线已被建立，变更提议应该针对需求基准线，并且应该涉及一个或更多涉众声援所提出的变更。每一个变更提议都必须被项目水平的变更控制委员会批准。变更需求 156
涉及软件架构的改变，有助于软件实现、测试或维护活动。一旦软件产品架构成为基准，变更需求必须被技术水平变更控制委员会批准。

每一次需求被允许更改，其对业务流程、功能和物理架构、说明、文档、测试计划和程序、工作分解结构和其他相关事项的影响都需要评定。这就会导致应用程序的功能不是起初计划的那样，并且会增加软件应用程序的质量或开发日程的风险。

8.5.4　评估项目可行性

软件需求相对于项目目标、工作包和风险的调整为理解项目成功的可行性提供了基础。这项任务的目的是确定在项目成本和进度的约束下，这个软件方案是否能完成。项目可行性是基于工作计划的精确性和与软件解决方案相关的技术风险。针对计划监测工作进程，会证实项目和技术计划是否精确地反映了执行的工作。通过启用原型工作来确定该解决方案是否是切实可行的，并且能够依照计划开发，可能会降低技术风险。

8.6　建立需求基线

随着需求的发展和成熟，需求基线应该被建立和保持，以减轻需求蠕变和基线的持续改变带来的影响。需求蠕变是指随着解决方案被建立架构和实现，产品需求被更改或补充的趋势。需求蠕变可能受涉众不断增长的“愿望清单”或者开发者意识到增加解决方案对用户吸引力的机会所驱动。

需求基线代表了设计的要求。需求基线不应该在设计解决方案完全成熟和完成之前建立，因为需求基线的变更会影响项目成本预测和日程的时间轴。这就是为什么需要通过功能分析和应用程序设计综合来探索软件解决方案，以保证需求集是完整的、一致的、可以实现的和负担得起的。

一旦软件产品需求基线被建立，所有被推荐的需求的变更都必须被项目变更控制委员会（CCB）正式批准。每一个提出的变更，包括恰当的说明和文档变更页在内，都应该被记录在工程变更提议书（ECP）中。

随着软件架构的发展和成熟，应该更新规范和文档树，以确定需求说明书、设计文档、工程制图和培训及用户说明书。表 8-2 给出了一个典型的软件需求文档项目，包含软件产品需求基线的列表。

表 8-2　软件需求文档

文档标题	类型
软件需求说明	说明
计算环境需求说明	说明
软件界面说明	说明
业务模型	设计成品
功能行为模型	设计成品
功能架构描述 • 功能组件说明 • 功能单元说明 • 功能界面说明	设计文档
物理架构描述 • 结构组件说明 • 结构单元说明 • 软件界面描述 • 软件集成策略	设计文档
需求跟踪矩阵	设计成品

157 ~ 158

软件需求管理

管理需求对软件开发工作的成功至关重要。需求定义了项目为满足涉众而执行的工作范围。软件架构的开发需要建立一系列达成共识的软件需求。软件需求必须是可控的，以保证软件架构和实现能够令人满意地通过验收测试和配置审核。如果项目初期软件需求不稳定，那么项目和技术方案、产品架构以及软件实现都要通过接连不断的变动来和需求的变化保持一致。

软件开发在软件解决方案从概念发展到可部署的产品过程中必须是可观察的。软件需求透漏项目取得成功的要素的同时，项目和技术方案揭露了操纵危险领域的危害。如果需求不必要、模糊不清或不稳定，项目就会进入混乱的状态。软件架构的开发在需求已经稳定和基线化之前不可能完成。

159 软件工程团队的主要目标是使软件需求、项目成本和进度目标互相对称。如果软件需求在开发的初始阶段不断变更，那么软件架构也会持续处于修改的状态。每一次软件需求的变更都会因为要求设计方案重做和重新计划而消耗项目资源。当部署管理实践尽力控制软件需求的变更时，变更是不可避免的，而且变更必须实现，项目才能带来令涉众满意的产品。

本章主要陈述了软件工程实践如何制定来控制需求的不稳定性，以及建立架构化的框架对包含那些必需的变更十分必要。

9.1 接受变更

相信涉众需求从最早的陈述开始不会改变来开展像软件开发这样技术驱动的项目是不可能的。涉众会因为以下两个基本的原因变更他们的需求：

1）他们对软件产品的理解会随着他们对软件开发工作的参与而提高。

2）他们的商业或业务环境会从软件开发项目被委任那一刻发生变化。

变更是不可避免的，因此它必须作为开发过程的完整要素来说明。为此，必须制定两个原则使开发团队能适当地管理不可避免的变更提议和请求带来的猛攻，并与之作斗争。首先，项目团队必须认识到时间是对手而不是盟友。每一次批准的变更都会造成资源的浪费，并且阻碍达成目标的进度。其次，由于一些变更必须被授权，项目结构和产品架构必须是可修改的以保证在未来几年后交付一个成熟的产品。

9.1.1 时间是一种宝贵资源

当一个团队着手一项新的软件开发工作时，对他们而言，最迫切的是认识到预定的交付日期即是最重要的动力。浪费资源去完成多余的或是不重要的任务是不可接受的。大部分企业通过各种流程和形式主义带给开发团队很大压力，使其从工程工作中分心。许多组织相较工程素质，更偏爱财务会计、法律和管理等方面的实践。充分利用企业资源的同时，必须保
160 证完成高质量的产品。否则，不管多少管理监督也无法完成最终任务。

技术规划必须考虑管理实践；然而，和这些实践相关的工作应该包含到每一项技术任务的定义中。工作分解结构及其工作包应该关注技术工作的识别。每一个工作包必须确定它为达成软件产品的临时开发目标所做的贡献。管理费用必须包含到每一个工作包定义中，而且出于资源方面考虑应该被约束，并作为企业的基本费用。记录技术规划、软件设计、实现和测试这些流程并不属于管理任务。

为满足涉众需求而提议的变更应该在花费许多工作确定它对项目框架的影响之前就对其优点做出评估。值得长远考虑的提议必须通过评估来理解变更对业务概念的重要性以及对软件产品架构和项目工作量的影响。提议的变更应该能在不增加产品复杂性和风险来满足项目费用和进度目标的前提下被整合到架构中。

软件产品架构包括以下 4 个基本的质量属性[⊖]：

1）完整性，是指软件要素分开实现但可以协作工作的能力（软件实现）。

2）可变性，是指架构可以适应变更（产品支持）。

3）可测试性，是指软件产品可以通过演示来查看是否满足其规格说明和涉众需求（验收测试）。

4）可用性，是指易于使用和培训最终用户。这些属性中的每一个都直接影响软件生命周期涉及的问题和整个生命周期成本。

通过评估期望的变更提议来决定该变更对以软件产品架构为代表的工程解决方案的影响。如果架构可更改，那么变更对其影响可能不会很大。假设架构和其文档一直更新，那么它应该相对简单以便确定那些必须完成才能对提议的变更造成影响的设计变更。这要求在软件架构元素和项目控制机制之间有广泛的可追踪性。

软件架构及其文档以及与工作包的关系有利于将提议的变更整合到软件产品和方案中。然而，每一次变更实现都会影响项目进度，因为员工工作会从既定的活动中转移到修正产品架构、文档和方案上面。在评估将变更整合到部分完成的架构描述中时，影响分析必须考虑 161
回归进度。原始的任务可能需要修正、重新安排或完全取消来适应变更。每一项变更会涉及对影响到已建立的架构设计文档的改写，对之前架构决策的再分析，以及一些附加分析来保证将提议的变更整合到现有的架构框架中。这可能会导致在变更加入软件架构的同时进度中断，以便为得到架构结论而建立一套新基准。

在最后分析阶段，如果产品架构将近完成，则不应该考虑采用变更提议。开发的架构定义阶段的初始规划估计应该包含足够的资源和安排空间来考虑需求变更。由于开发早期阶段有足够的灵活性，因此许多重要的变更提议可能会得到满足。每一个工作包（如缓冲区）应该包括既定的时间表来保证关键设计审查（critical design review，CDR）里程碑和向软件实现的过渡得以实现。

9.1.2　变更影响分析

每一项提议的变更必须通过分析来决定变更是否应该予以授权以及整合到开发框架。实施一份详细的影响评估之前，必须确定已提议变更的关键程度。已提议变更的关键特性应该能从软件业务适用性角度表明变更的必要性。表 9-1 列出了和标准配置管理工程变更提议的

⊖ 参见 *http://www.softwarearchitectures.com/go/Discipline/DesigningArchitecture/QualityAttributes/tabid/64/Default.aspx*.

优先级编码相关的关键程度的常见产业层面。这些关键程度旨在为那些可能会提高企业投资回报的变更提议确定优缺点。是否决定开始一项变更影响评估应该基于涉众对变更重要性的感知。

表 9-1　软件变更提议关键程度水平对比

基于管理的工程变更提议（ECP）配置优先级编码	软件变更提议的关键程度水平
紧急的：会影响到那些如果无法按时完成、可能严重危害组织安全的业务特点变更	紧急的：会影响到那些如果无法按时完成、会严重损害涉众业务效率的业务特点变更
改正可能造成严重的人员伤害或设备毁坏的危险情况	改正可能造成严重的人员伤害或设备毁坏的不安全业务情况
紧迫的：会影响到那些如果无法按时完成、可能严重损害设备、软件或人力的任务效率的变更	必要的：会影响到那些如果无法按时完成、可能严重损害设备、软件或业务过程的业务效率的变更
改正可能造成人员伤害或设备毁坏的潜在危险情况	决定性的：可能影响到对扩大产品对潜在顾客吸引力非常重要的变更
满足重要的合约需求	遵从的：满足契约或合同需求
常规的：当紧急或紧迫实现不适用、不需要或不合理时	建设性的：可能影响到如果没完成，则不利于产品可行性或扩大产品对潜在顾客吸引力的变更
	主观的：会影响到那些支持一个或多个涉众业务过程的变更

变更影响评估是指成本表明工作包对整合变更的必要性，收益代表那些涉众认可的利益的成本效益分析。无形的收益可能很难量化，但未来的商业机遇、技术经验和因承担新需求所收获的能力，以及对企业声望的影响也可看作优势。一份变更影响评估包括以下步骤：

1）变更网络分析。确定那些会被需求变更影响的相关技术任务。如果需求可追踪性关联已全面确定并维护，那么对现有需求的变更影响到的相关技术任务可以相对简单地确定。新的需求要求建立可追踪性关联来确定相关任务，这些任务必须做出评估来决定提议的变更对技术方案的影响。新的需求可能会给工作计划带来额外的任务。这包括将变更整合到软件架构、文档和技术方案（尤其是软件实现和测试方案）中的任何必要性返工。

2）冲突评估。确定那些可能会和提议的变更发生冲突的需求。需求可能规定不一致的、有分歧的或矛盾的设计目标，而这些目标必须在指定需求前决定。对冲突需求的调整可能会造成额外的技术任务返工和进度回退。

3）方案可行性。建立能满足提议的变更和相关联的冲突设计目标这样的架构化解决方案的能力可能会给项目目标的实现带来额外的风险。这往往导致建立了一个能在冲突的设计目标间达到令人满意的折中的架构化设计。权衡分析和设计方案的风险评估对派生一个可行的并且极为有利的设计方案是完全必要的。

162
/
163

4）方案成本评估。确定将提议的变更整合到工作计划中的预期成本。这包括现有工作包的修改和将额外工作包列入工作计划。方案成本评估应该确定那些为实现变更提议所需的资源上的变更，以及任务调度和里程碑式成就的修改。

5）认知收益评估。确定由已提议变更的授权带来的预期收益。预期收益表现在涉众满意度、产品可行性、市场增长潜力和从累积的经验中获得潜在商机等方面。在合同安排中，客户应该为整合有益的变更提供所有相关成本。如果成本过高，那感知到的收益不会超过变更的成本。

9.1.3　调整项目里程碑

软件开发进度在软件产品从概念发展到业务产品的过程中必须是可观察的。开发里程碑代表产品从一个开发阶段过渡到另一个的演化过程中那些重要决策点。在每一个里程碑，应该对技术成果进行审查来确保产品开发按计划向完成项目目标前进。里程碑代表技术性工作的间断，因为要决定当前的产品定义是否足够完整来保证下一个开发阶段的启动。

每个阶段的目标都可能会受到已授权的变更提议的影响。项目里程碑不应该因为变更请求而推迟，除非变更的数目过大。在每一个里程碑，应该对将每个变更请求整合到软件架构中的状态进行审查来了解完成工作估计的时间。这会导致里程碑完成的推迟，而且完成里程碑的标准也应该予以修正以便反映出完成将授权的变更请求整合到软件架构中的需要。这有效地将计划的开发阶段的完成日期延长到晚于里程碑评审。图 9-1 展示了计划的阶段相关的进度表和里程碑以及授权变更请求可能对完成标准的影响。

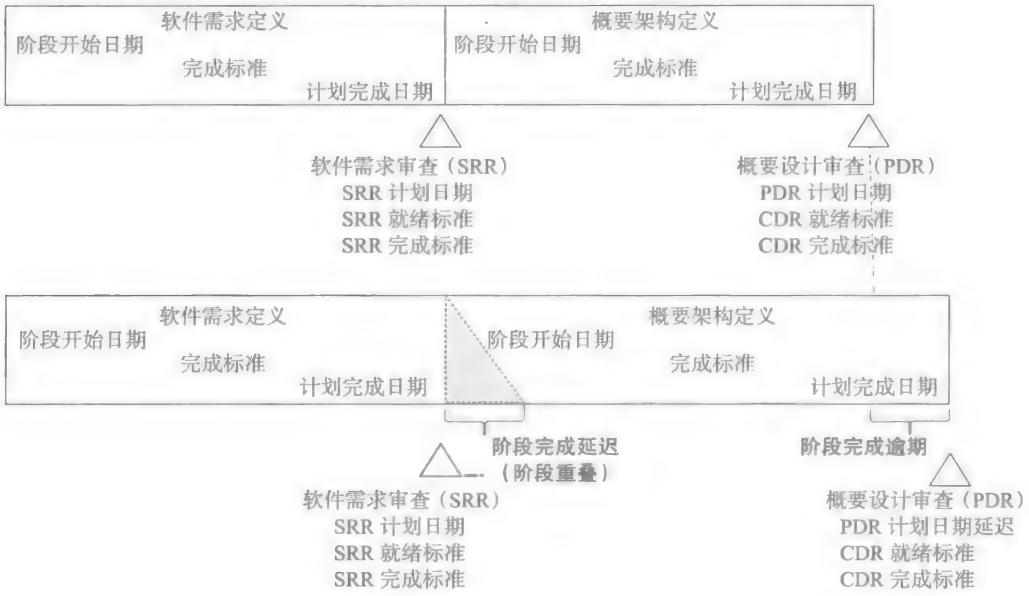


图 9-1　阶段相关的进度表和里程碑

每当延迟发生时，阶段完成日期和相关的里程碑评审之间的关系就会引起混乱。每一个开发阶段通过确定开始日期、持续时间和计划的完成日期建立计划。必须建立一系列标准来决定一个阶段什么时候完成。里程碑审查通常与每个阶段的结束日期一致以便开发工作进度的技术性审查。这些审查旨在确定那些项目是否应该开始下一开发阶段的决策点。每一次审查由一系列就绪和完成标准所定义。就绪标准确定了执行审查必须满足的最低条件。审查过程中，可能会指定许多影响到产品开发工作当前状态的活动。这些活动中的一些必须优于审查完成。因此，审查可能直到活动项满足才会成功完成。大多数情况下，上述局面会造成两个原定逐次执行的开发阶段的重叠。

对项目进度而言，受进展而不是计划日期驱动很重要。阶段启动的计划日期可能不会取决于之前阶段或审查的成功完成。软件架构中可能存在稳定的，而且可以在里程碑审查中的活动项解决时移入下一个开发阶段的元素。挑战是找到架构的什么元素会被活动影响，因此就不用花费工作定义或设计那些可能会变成活动结果的元素。

为保证开发阶段顺利进行，必须更新计划来确定活动对阶段相关的工作包造成影响的范围。通过将工作包重组的方式充分利用人力资源可能会减小上述影响。再规划工作的目标是确定计划的里程碑审查日期是否能不管变更提议或由审查生成的活动而达成。

164
/
165

9.2　明确需求

明确软件需求包括对业务和商业过程中涉及的软件产品角色细致全面的检查。需求陈述可能看起来简单易懂，但可能暗藏许多需求规定的实现和测试条件相关的困难。有 9 项原则可以应用来保证需求精确地表述，在既定的项目约束内满足，并且不会反过来影响开发后的过程或生命周期成本。这些原则是：

1）需求表述了产品必须做什么和如何运作。每一项需求应该确定业务功能及其相关的有效性度量方法。在软件产品层面，需求应该说明软件产品计划执行的实际功能。每一项功能必须在性能的可接受范围方面是可度量的，因为这些性能是产品获得涉众认可所必须的。在较低层面，需求应该说明产品设计的单个元素。低层需求要说明数据处理流程方面的功能，而且必须确定那些保证产品执行的必要性能的可接受范围。

2）需求必须清楚明白。需求不可以使用含糊不清的语言来表述。一份陈述适当的需求应该写出来有且只有一种解释。一份清楚明白的陈述是明确的（使用清晰明显的方式说明所有细节，想要表达的含义毫无疑问），毫不含糊的（没有任何疑问和误解），清晰明白的（明显不同于其他）。需求陈述所用的自然语言从根本上说是不确切的，因为单词会有多个含义和内涵。因此，需求分析任务需要保证每一个需求被适当地陈述来消除误解或得到其他分析文档支持的解释。

3）使用设计模型来说明概念，消除潜在困惑。通过建模向涉众和开发团队的成员说明设计概念。模型可以用来说明产品行为（功能和性能）和特点。模型可以是静态的（如图表）或动态的（如可执行文件、原型、仿真），也可以在必要时更详细地解释需求的含义。

4）需求不应该强加不必要的设计或实现约束。约束会限制开发团队的自由和解决方案的空间。解决方案考虑的需求应该作为弄清涉众期望的建议。

166

5）需求如果过多应该受到质疑。涉众需求可以通过质疑需求的业务或商业要求来理解。这些质疑如果被证实，会增进开发团队的理解，也可以弄清那些从最初的协商中推断出来的假设。涉众会对一些软件操作（也就是一系列复杂的需要较高敏捷度和技能的精神或身体操作的表现）的复杂性抱有过高的期望和很少的欣赏。许多需求在涉众意识到他们夸张的要求后会被简化或降低。

6）需求会迫使成本和进度间产生冲突。每一项需求包括与产品开发、开发后的运作和产品支持相关成本。应分析需求，以理解每一项需求的生命周期成本影响。应该有其他表述需求的方式，可以降低软件开发面临的风险。

7）需求可能会加大设计复杂性。需求可能会以暗示或放大产品复杂性的方式表述出来。降低产品的复杂性是一项可以减少产品开发和维护成本的同时提升使用便利度的必要原则。应该执行权衡分析来评价可选的需求说明以便决定对设计复杂性的影响。

8）需求不应该放弃对接口定义的控制。每当软件产品必须与其他系统连接时，不管是业务上的还是正在开发的，软件开发团队必须参与接口的定义。每当软件开发团队对控制范围做出妥协时，他们合理计划开发工作范围的能力就会受到影响。外部系统可能会老化，需要技术更新或重新开发。授权接口控制前了解连接系统的寿命十分重要。重新定义接口可能

带来软件性能上显著的提高。

9）风险总是和需求相伴而生。通过对需求的评估来确定完成项目的潜在风险。需求是风险的源头，需求所带来的风险也应该在需求被接受前做出评估。包含较大风险的需求应该简化以使最初交付的软件满足成本和项目约束。需求的设计和实现在不危害项目成功时可以并行。可以通过回答下述问题对需求做出评估：

- 需求能否在不消耗过多技术资源的情况下得到满足？
- 如果无法满足需求，会产生怎样潜在的结果？ 167
- 为了试图满足需求可能出现的最好、大概、最坏情况是什么？
- 如何重述需求可以降低项目成功的风险？

9.3　需求分解和分配

最初的软件需求通常是一组庞杂繁复的条款，以验证最终交付的软件产品质量。每一项需求必须从自然语言的形式变换成可实现的软件产品的一系列设计特性、特色或质量属性。这个基本前提是本书所描述的软件工程与现有的软件开发方法或实践的区别所在。从需求到设计的变换要求将软件工程实践应用于设计、计划、实现和维护软件产品。从软件需求开始，技术团队必须精诚合作，建立一套可以实现、测试、维护的软件产品架构。需求在产品架构设计、测试用例和流程、技术方案、控制机制中必须是可追踪的。可追踪性有助于技术团队对新提出的变化进行响应以满足涉众需求、软件需求或者设计挑战。图 9-2 给出了需求分解和分配的流程图。

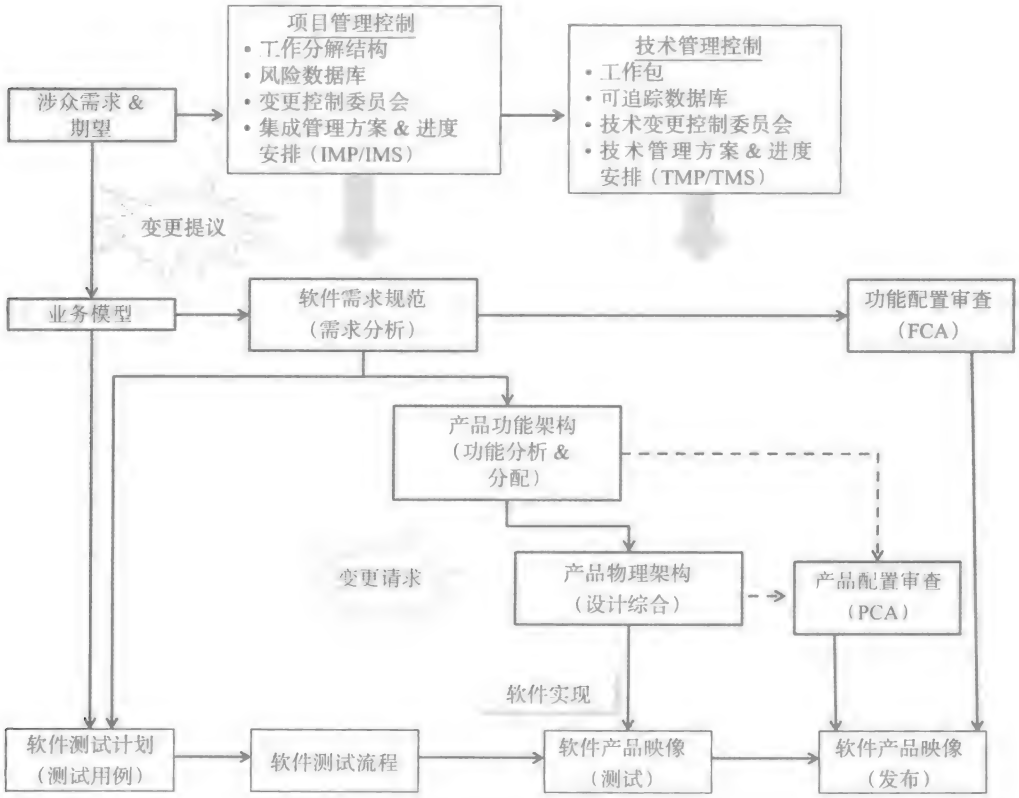


图 9-2　需求分解和分配流程图 168

需求被进一步分解，以明确较低层次的功能性和性能方面的需求，它们是满足原始需求所必备的。以需求分解为基础，可以逐步求精地明确软件产品如何执行它的业务功能。面对大而复杂的功能，需求分解可以将其变成一系列基本功能，而这些功能可以用来开发结构化设计概念。需求分解的必要性表现在两方面：一方面是检查需求所定位的问题空间；另一方面是决定使用串行还是并行功能，这些功能的执行是为需求提供适当的软件解决方案。

9.3.1　功能分析

业务或商务需求难以直接分解为软件结构元素（比如，类）。需求必须分解为子需求，使得这些子需求的设计方案直观易懂、没有疑义。因此，软件产品的业务需求必须首先分解来发现那些中间的、根本的功能（组件和单元），这些功能是软件产品必须完成的。

功能分析是指这样的软件工程实践，像软件需求中描述的那样，把业务功能分解为过渡性的（功能组件）和基本的（功能单元）功能。由此产生的功能架构确定和详述了一整套设计解决方案必须包含的功能。该项实践强调自上向下的方法，即将问题分解为一整套设计解决方案必须表现的功能。然而，该实践需要反复迭代功能性的层级来应对复杂功能挑战。

9.3.2　性能分配

功能的组合必须满足和业务需求相关的性能要求。因此，功能性解决方案必须满足业务性能特点，后者建立在业务需求之上。功能性解决方案的功能时序和资源利用必须满足软件产品指定的性能需求。

随着业务功能的分解，性能分配涉及性能预算的建立。通过运用计算机科学、数学和计算环境性能特点方面的知识，分析核心功能以决定执行性能方法的范围。功能性能预算必须被修正，以便为功能架构的每一个元素建立功能规范。当功能组件和单元的功能规范可以被验证为满足软件产品需求时，该功能架构可以被认为是完备的。

169　贯穿于功能架构的性能需求分配为达到软件产品性能需求奠定了基础。软件性能工程要求响应时间、吞吐量和资源优化水平满足指定的性能目标。软件性能依赖于计算环境特征。在软件架构定义过程中必须考虑的计算环境特征包括（但不局限于）如下：

1）执行时间

2）内存优化

a. 初始内存（随机存取存储器）消耗

b. 虚拟内存（辅助存储器）消耗

3）交换时间（虚拟内存管理读写延迟）

4）数据存储延迟（在存储器中访问特定位置的时间）

5）数据存储吞吐量（数据从存储器中读写的速率）

6）中断延迟（设备引起的中断与设备服务之间的时间）

9.3.3　结构化单元综合

设计综合是指用于建立设计解决方案或物理架构的结构化单元的软件工程实践。结构化单元代表软件产品中的基本块，同时结构化单元可以促进软件实现（编码和测试）。通过组合相似的功能性单元和解决功能规格之间的差异，可以得到结构化单元，并由此得到每个结构化单元的集成规格说明。在软件实现过程中，大部分编码是在单元水平开发完成的。

9.3.4 结构化组件综合

通过发现那些需要集成到一起以提供中间结构化装配的结构化单元，可以确定结构化组件。结构化组件代表一系列提供递进功能的软件产品的那些增量的装配、集成以及测试。结构化组件为软件组件集成和测试建立了策略，以保证它们能在软件实现过程中完成。

通过发现在集成较低层次的结构化组件或单元中出现的功能性的和性能方面的特点，可以指定结构化组件。在软件实现过程中，结构化组件可能需要生成额外的代码来管理执行控制逻辑。这导致在组件集成和测试过程中必须识别内部和外部接口。

9.4 需求可追踪性

软件产品需求在整个功能和物理架构过程中必须是可追踪的，这有利于促进涉众的变更提议、设计变更请求和产品配置审核。这些架构化的观点（功能和物理架构）确立了软件产品设计（指架构化设计），它显然不同于在软件实现过程中修正的结构化单元编程设计。软件实现过程中，利用计算特定语言的构建将每一个结构化单元规格转化成清晰的设计。通过程序化设计来实现计算语言指令的软件单元，这些指令是指特定的实现语言（如 Java、C++）。图 9-3 展示了从涉众需求到软件实现单元和组件过程中，需求是如何被追踪的。

170

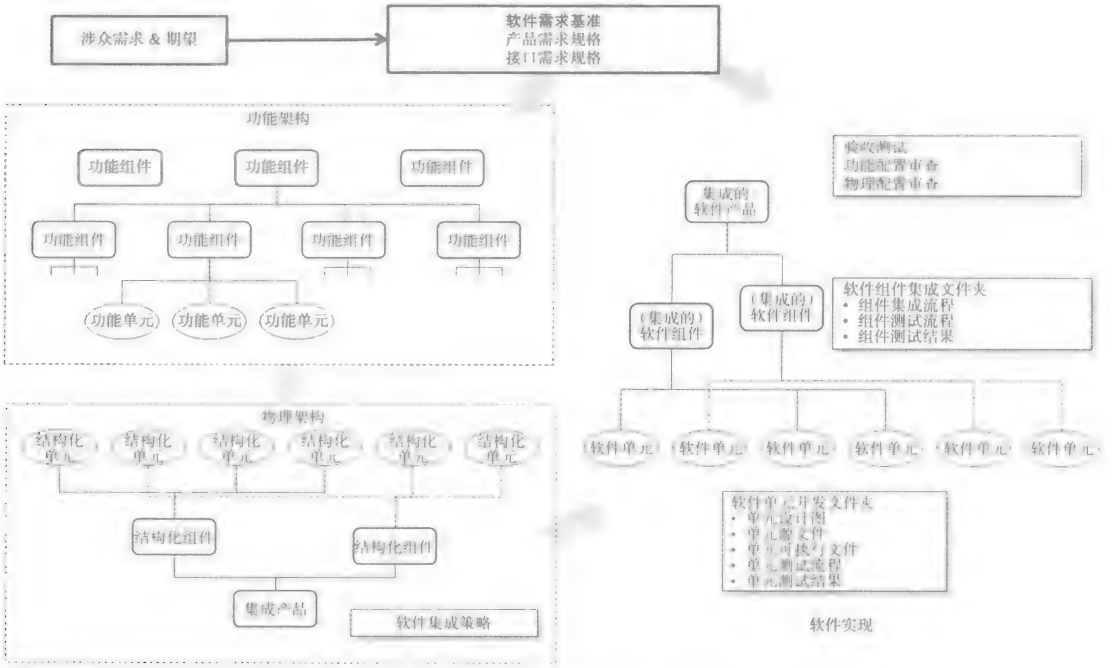


图 9-3 软件产品架构内的可追踪性

在涉众需求、软件需求、架构、实现和测试工件间建立可追踪性是很必要的，以使开发团队能对提议的变更及时响应。可追踪性有助于评估提议变更的潜在影响，而且对保证软件开发项目达成目标十分重要。

9.4.1 变更控制

软件开发项目内存在正式和非正式的配置管理实践会影响处理变更请求和提议的方式。

为了简化变更控制的概念，变更提议应该考虑到那些需要额外项目资源整合到项目和技术方
171 案中的提议变更。变更请求是指影响软件架构或实现的设计变更，而且是实现特定软件需求必须要考虑的。

项目变更控制委员会有责任授权变更提议。这要求提议变更对项目和技术计划的影响应该使需求可能影响到的重要涉众了解。技术变更影响陈述必须保证提议的变更在增加的项目成本和调度资源范围内被满足。如果项目成本和进度目标保持不变，商讨淘汰还是适应其他软件需求是有可能的。

技术变更控制委员会有责任授权那些影响之前特定的软件产品架构元素中变更的变更请求。变更请求代表对那些利用软件架构元素解决障碍、复杂性和晦涩等问题的软件架构定义的建设性修改。变更请求应该简化软件实现和测试工作或解决架构化规格缺陷和不精确性问题。

9.4.2　配置审核

基本上，需求可追踪性对支持功能和物理配置审核极其重要。软件架构提供了软件产品配置到其实现、测试结果和文档工件的映射。配置审核应该优于软件部署或发布执行以确定最终软件产品：

- 满足特定软件需求。
- 包含授权的变更提议和请求。
- 172 在配置控制下通过精确的设计文档、操作手册和源文件为软件维护做好准备。

制定功能架构

本章主要描述了功能架构和其多种表现形式。它还提供派生功能架构的准则。第 11 章为将软件需求规约转化为功能架构所需要的功能分析与分配实践建立了一系列详细的任务。功能架构为从软件产品中继承结构化配置和物理架构提供了基础。物理架构包括那些表示软件产品的结构化配置的文档、图纸、图表等。在这些任务中，从出发点就确定与其他软件工程任务的联系，这些任务有需求验证、评估设计复杂性和风险的软件分析、软件设计综合和架构控制。第 11 章中提供的详细任务描述中可以发现这些联系。

10.1 功能架构的动机

功能架构提供了一个没有物理特点或结构化特点的软件产品的工作视图。这是从明确了软件需求的业务或商业模型中派生出来的。最上层确定了与外部实体交互的主要软件功能，即描述软件对外部刺激的反应。通过分解主要功能来与涉及软件产品必须提供的数据处理服务相关的额外的细节。图 10-1 说明了功能架构作为软件需求向软件产品设计转换的第一步所扮演的角色。

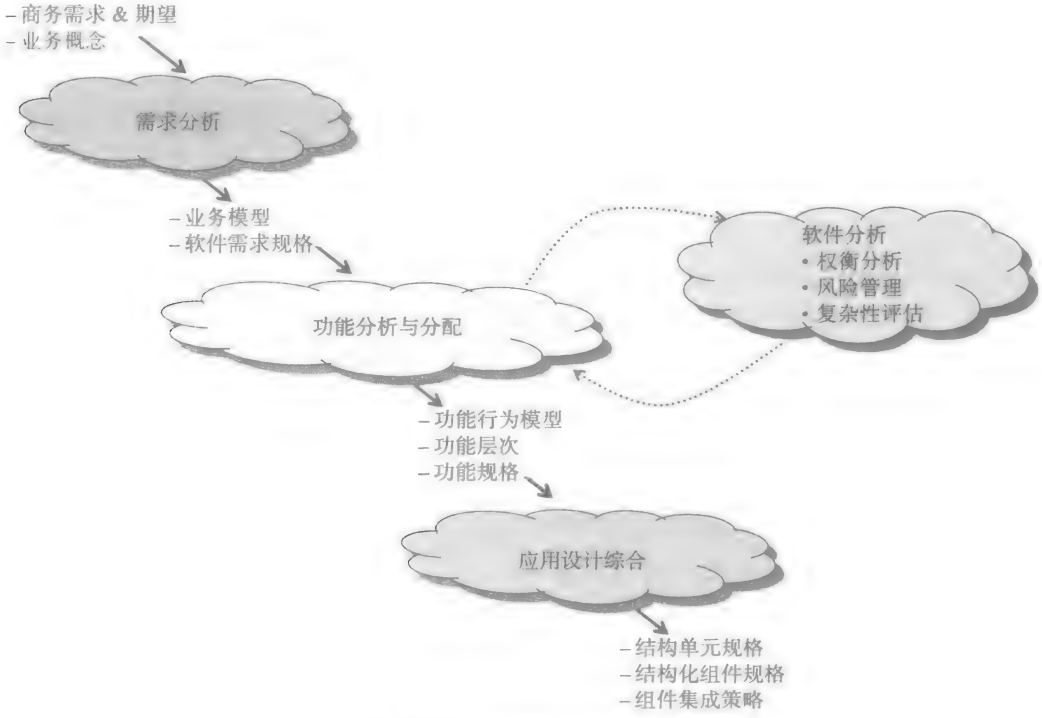


图 10-1 功能架构的角色

功能架构代表结构化设计软件产品的用意及使用。业务模型描述执行商业或业务过程中软件产品的角色的同时，功能架构解释了软件产品必须执行的数据处理动作。功能架构最终必须被分解为调用时只产生一个结果的基本功能。基本功能即功能单元，而且必须明确以支
174 持软件产品的结构化设计。

功能分析与分配实践为软件需求向软件产品必须具有的功能事务的转换提供了方法。功能架构代表一系列综合的、集成的数据处理事务。通过功能分析与分配来实现以下 6 项基本的涉及软件产品质量的设计挑战。

1）制定一项解决方案。设计解决方案需要许多单个软件元素协作来支持商业或业务过程。这包含大量数据处理事务、服务或行为线程。软件数据处理事务包含许多难以理解的分析性的排列组合。功能分析为确定大量可能的软件必须促进的数据处理序列提供了方法。功能分析与分配实践系统地识别并调查那些离散的事务线程来彻底地明确功能解决方案。

2）明确歧义。用来沟通以及表述需求的语言包含不严密的、模糊的、不清楚的、含混的、不严格的、朦胧的单词和词句。功能分析保证每一项功能清晰明确，因此功能解决方案的描述中不存在误解。

3）解决假设。所有的假设都必须在解决方案定案前同涉众一起解决。那些没有被质疑与解决的假设可能导致软件产品无法有效地满足消费者的需求和期望。功能分析也可以用来推测和评估假设，从而消除那些作为产品设计基础的判断或意见。功能分析强调不完整或不确定信息的存在，以此引起对未证实的推测的注意力。

4）达到性能目标。明确的性能需求或目标必须被理解，而且设计的软件产品也要达到标准。利用软件设计技术、图纸和模型展示软件性能特点。必须分配软件产品级别的性能度量来提供低级别的设计愿望。最初的性能预算必须确定，才可以对设计策略进行评估。一旦解决方案通过评估认为合适，就可以为设计元素分配和明确性能需求。

5）确定资源利用率。数据处理的效率和效果取决于计算资源利用率的规则。功能架构通过对资源利用率建模使得设计在有效执行方面得到优化。软件设计必须对资源利用率对性能目标的影响保持敏感。

175 6）简化解决方案。任何软件产品的复杂性都无法让人满意。用户界面设计和用户交互不能让人费解。这要求大量的用户训练，也可能劝阻潜在的客户以机构标准接受软件产品。设计复杂性直接转化为代码的复杂和晦涩。软件维护成本会因设计复杂性成比例增加。功能架构提供了最初的可以调整复杂性的设计范例。

功能分析与分配的目标是制定一个完整的、一致的、验证的功能架构，使其符合明确的软件需求。软件架构必须确定每一项数据处理任务，包括故障检测、整改方案和业务模型的相应退化。当每一个功能组件、单元和接口都明确后，功能架构才是完整的。功能单元和接口即被用来派生软件产品物理架构的“构件块”。物理架构识别并明确了软件实现过程中会被详述（设计、编码、集成和测试）的结构化软件元素。第 12 章描述了物理架构以及它如何从功能架构中派生而来。

10.2　功能架构本体论

本节确定那些用来描述功能架构的术语。根据 Dictionary.com 中的说法，本体论为“通过事物的本质为其划定不同范畴而形成层次结构”。因此，接下来的“事物”即用来描述和记录功能架构的实体。这些标签并非软件工程特有，同样适用于描述人造产品的功能。

10.2.1　功能组件

一个功能组件代表一项复杂的、软件产品必须执行的任务。功能组件会在控制被转移到组件中执行时被激活。每一个功能会将输入或全局或局部变量形式的一或多个数据项转换成输出数据项或中间变量。功能复杂性在下列任何情况存在时会很明显：

- 一个功能包含几个数据转换动作，且至少一个动作没有清晰简单的解决方案。
- 一个功能包含可辨别的条件响应。
- 一个功能包含多个与其他功能或外部系统、用户或其他软件应用（如数据库）的接口。

功能复杂性迫使解决方案进一步分解为较简单的功能组件。分解要求功能组件划分为两 176
个或多个子功能。一个功能被指定为一个组件表明它包含更低级别的功能细节来明确地表达执行数据处理或转换的方式。要得到一个简单的解决方案，几层分解是必不可少的。

10.2.2　功能单元

功能单元代表层系最低层的元素。功能单元表明不需要进一步分解来表述解决方案了。功能单元在功能的设计或实现容易理解时会被发现。功能单元应该执行单个简单的任务；应该从若干有限的资源中收到作为输入的数据项；应该将数据处理结果输出到若干有限的接收功能或外部元素中。

10.2.3　数据项

数据项代表软件产品必须处理的信息。数据项可能本质上很复杂，因此需要分解来表述涉及数据处理的特定数据元素。例如，一张借记卡使用 ATM 读卡器可以读取的账户信息进行编码。最初的数据项可能就概括为“账户信息”。然而，精确表述借记卡上各种编码信息是十分重要的。典型的客户信息可能包括客户姓名、卡的有效期、银行路由号码、检查账户号码和储蓄账户号码。这些信息元素中的每一个都必须在功能分解的某个级别作为数据项被识别来表述数据规格。

数据项包含可能影响软件产品性能的特点。数据有一个大小属性，表示数据项代表的信息数量，该属性可能影响存储、检索和传输速率。传输速率影响在两个功能间传输数据项需要的时间，这受限于计算环境的内部总线子系统，因为它通过外部接口在计算机内或计算机间完成组件间数据的传输。外部接口的容量也会影响数据由主计算设备向外部系统的传输速率。

10.2.4　功能接口

在功能间交换的数据代表功能接口。功能接口为要交换的数据确定需求。功能接口为建立功能接口规格提供了最初的基础。它也确定并描述了在内部软件功能与软件产品的相互作用间通信的数据元素。 177

10.2.5　外部接口

在软件产品和外部系统、设备、用户或其他软件应用间交换的信息代表外部接口。外部接口应该在软件需求分析阶段就明确，并且在功能架构时解决以便支持接口设计。

10.2.6　控制结构

控制结构为指导执行流提供了方法，以便执行数据处理任务，这解释了对数据处理中间结果的条件性处理。从业务模型的角度，控制结构代表的是商业规则或决定过程如何执行的业务规程。在功能架构中，控制结构代表的是决定数据处理执行如何继续的决策或计算逻辑。一般的控制结构如下：

- 分支——包括一连串的数据处理任务或功能序列的执行路径。
- 并发——使得多个行为或分支线程并行开始和执行。
- 选择——基于某种条件论证，使得一个行为线程开始。
- 迭代——使得一个行为线程重复一次或多次。
- 触发——激活一个动作、过程或一系列事件。

这些控制结构具有大部分计算机语言实现的类似构造。然而，为了建立功能架构的目标，坚持特定实现的控制构造并不可取。在 Ada 和 C++ 编程语言中，并发与任务同步是类似的。在大部分编程语言中，选择表示 If (If…Goto, If…Then, If…Then…Else) 或 Case 语句。大部分编程语言中，迭代表示类似 For…Next, Do While, Do Until 或 ForEach 这样的循环机制。

10.2.7　资源

资源表示任何可用性可能影响软件产品性能的可量化实体。资源利用率影响软件产品的性能评估，而且一般包含动态的随机行为。资源可以用来理解计算环境特征，它们可能影响执行计时或其他阻碍数据处理有效性的资产。资源有几个特点可能影响软件性能：

178

- 库存容量。确定任何时候都可用的单元最大数目的容量。
- 库存。可以用来支持软件产品的数据处理需要的资源总量。
- 消耗量。功能可能消耗资源，拉低库存，因此减少其他功能可用的资源数量。功能也可以通过补充资源来体现对库存资源的补充。
- 捕获量。一个功能可能会获得重用的资源来支持其执行，并且在功能执行完时释放资源。
- 再补给量。功能重新补充的消耗资源的数量。

10.2.8　数据存储

数据存储是指能保持数字数据，并且支持数据持久保存的仓库。数据存储支持数据储存和检索功能，这些功能包含搜索并操作数据记录的事务。围绕影响软件性能的数据存储事务的问题包括数据存储可用性、事务处理和回滚、数据模型定义、数据安全和评估控制以及数据库查询优化。

10.3　构想功能架构

功能架构需要回答以下问题，“软件产品必须执行什么功能（数据处理任务）来满足既定的软件需求？”这是一个在没有解决模块、子例程、对象和软件轮廓的其他物理形式的结构安排情况下，从可用信息中搜寻解决方案的过程。这可以通过对功能、数据和控制流的多层细化或分解来实现，这导致产生两个必要的视角：功能层次和行为模型。

行为模型为软件功能提供了比功能层次更精确的描述。行为模型中识别的复杂功能应该

被分解成描述每个功能如何执行的单个模型。行为角度上功能的分解可以用来生成功能层次。然而，在行为建模实践的同时开始功能分析不是必须的。软件需求中派生的功能经过系统的分解，可以在分析功能行为前建立功能层次。分解和行为建模这两项实践互补，应该被用于明确地描述功能解决方案。

准备功能架构的方法包括以下 5 步：

1）*继承基本功能*。通过对软件需求的分析来确定软件产品必须执行的基本功能。由于 179
软件需求是从业务或商业过程角度来描述软件产品，基本功能的确定可能和需求不一致。这可能是用来描述业务或商业过程的典型术语。软件需求必须被转换成那些反映相关领域语义的软件功能。一个软件领域反映了这样一个研究领域：定义了一系列为解决领域内某个问题而构造的任何软件程序的普通需求、术语和功能。

2）*分解基本功能*。由于基本功能是从业务或商业过程描述中得到，可以假设基本功能比较复杂。每一个复杂的功能代表一项因解决方案太具挑战性而难以理解的软件任务。功能分解涉及通过研究问题空间降低复杂性，以便提高有助于构思设计解决方案的理解。

- *功能分解*。功能分解需要确定完成复杂功能所需的一系列操作。大多数情况下，完成一项复杂的功能有多种方法，而且每一种设计方法或候选方法会表现出不同性能和架构化质量特点。
- *功能行为建模*。通过确定功能序列、数据项和控制机制创建一个功能如何表现的模型对执行源功能十分必要。源功能既定的性能特点和资源利用率标准必须在子功能间做出预算与分配。
- *候选方法评估*。每一个功能分解或行为模型都包括设计选择。相互竞争的候选方案应该利用软件分析活动来评估、优化并选择一个推荐的解决方案。为满足既定的性能需求和期望的软件质量特点，应该在有效性、适用性和风险方面对每一个解决方案进行评估。
- *隐含行为识别*。通过评估功能解决方案来识别那些能增强数据处理彻底性的推断的、互补的或补充的行为。应该假设规格说明不完整，并将数据处理动作聚集成一个合适的表达式。通过对功能解决方案一丝不苟地评估，确定为应对数据处理过程中可能发生的任何情况所需的行为。
- *功能解决方案优化*。通过评估功能解决方案，确定那些会对软件性能和资源利用率有很大影响的方面。最初的功能解决方案中重要的方面应该十分精炼，以便增加解决方案的有效性，如数据完整性、失效条件的可能原因、输入 / 输出数据项定义和数据转换算法的精度。 180

3）*明确解决方案*。必须明确由复杂功能分解得到的功能解决方案的那些元素。每个元素的需求必须为每个子功能建立性能和资源利用率目标。数据持久性事务特点、故障检测和恢复动作以及数据项属性都必须量化。功能解决方案的规格说明为剩余的复杂功能的进一步分解、行为分析和规约提供了基础。

4）*评估功能复杂性*。通过对功能解决方案的评估，决定进一步的功能分析与分配是否必要。

- 如果一个功能确定复杂难懂，那么对于每个复杂功能重复 2）、3）步。
- 如果一个功能确定较简单，那么它的功能说明和层级关系和行为图表必须放在技术配置控制之下。

5）简化功能架构。像功能架构演化一样，评审功能配置时必须谨慎以便确定架构得以简化的机会。

10.4　记录功能架构

功能架构包括一系列必须准备的来记录功能架构的软件工程工件（图表、模型和功能说明）。虽然许多软件方法规定各种各样的图表，碎片化表示的数目无法提供一个完整的、一致的软件架构视图。这一点挑战了涉众和软件开发工作成员的能力，使其意识到解决方案的整体性。

功能架构的两个代表：功能层次和行为模型对表述整个解决方案十分必要。另外，有4种额外的表示来支持功能解决方案的分析和规约：功能时限、资源利用率概述、功能规约和需求分配表。以下小节将在目的和特征方面描述每一个工程工件。

10.4.1　功能层次

功能层次传达了软件规约向软件产品必须执行的业务或商业过程的基本单元（功能单元）
181 的转换。这提供了直到最初的软件产品设计配置时软件需求的可追踪性。

功能层次在以下三方面提供了软件产品复杂性：

- 分解的层数。
- 分解的每层的广度。
- 派生结构化设计的基本功能元素（功能单元）的数目。

分解的层数在整个功能层次中不一定始终如一。基本功能包含的分解层数很好地反映了那些执行的数据处理事务的复杂性。分解的每层的广度暗含着软件组件集成和测试过程中可能遇到的问题的复杂性。功能单元的数目表明了软件单元设计、编码和测试活动的范围。然而，这些指标会作为软件设计综合活动的结果被牢牢确立。软件设计综合过程中，普通的或紧密耦合的功能单元可能被组合成单一的结构化单元，因此会减少软件单元设计、编码和测试的预期工作量。

10.4.2　行为模型

行为模型提供了支持功能组件规约和单元规约的决定性信息。它以减少猜想、假设或推测的一种标记法将软件行为表述成一种完整、精确的表示。行为模型可能是数据处理事务的软件产品执行的静态或动态模型，而且确定了对潜在的数据转换结果、操作失误和硬件故障的响应。

行为模型对软件工程活动而言至关重要，因为它为那些多种多样的必须执行的数据处理事务提供了清晰的表达。行为模型为使得设计解决方案得以评估和解决的软件设计建立了基础。它提供了制定功能时限和资源利用率概述的设计框架。功能行为模型描述了：

- 必须执行的功能序列（功能流）。
- 软件产品与外部系统、应用或操作者之间的数据流（数据流）。
- 功能或功能接口间数据流（数据流）。
- 确定条件功能序列间执行流程的商业规则或控制逻辑（控制流）。
- 完成每个功能必须的资源（资源利用率）。

182 控制逻辑可能与功能的结果紧密相关，并且会建模成在多个功能流序列中在选择处终止

的功能。接下来要执行的选择的行为路径可能很确定地强行选择一个期望的序列，或有概率地随机选择一个序列去执行。行为模型可能包含一个概率分布方程来代表随机选择。下列为一般的概率分布功能：

- 伯努利分布（Bernoulli distribution）
- 二项分布（Binomial distribution）
- 均匀分布（Uniform（discrete）distribution）
- 泊松二项分布（Poisson binomial distribution）
- 几何分布（Geometric distribution）
- 对数分布（Logarithmic distribution）
- 指数分布（Exponential distribution）
- 帕雷托分布（Pareto distribution）
- 卡方分布（Chi-square distribution）
- 韦伯分布（Weibull distribution）

可以利用一个可执行的行为模型来支持设计权衡分析——软件分析实践的元素（见第 14 章）。一个可执行的模型提供了软件产品的工作原型而不是依靠抽象的图表。通过比较竞争的设计候选方案来决定哪种设计方法在预期的业务条件下表现最好。Monte Carlo 仿真可以通过将模型封闭在一个循环控制结构中以及对功能时限和资源利用率的重复分析来实现。这使得通过可以被用来指导软件测试用例开发的模型，对关键路径的概率识别成为可能。可执行模型在验证功能架构是否满足既定软件需求方面也很有价值。

10.4.3　功能时限

功能时限为执行每一个业务场景或行为线程所需的时间提供了一种评估方式。时限描述了执行过程中发生的每一项动作的持续时间，而且使得对时序要求严格的设计需求的分析成为可能。执行时间的变化可以通过生成的那些影响商业规则或控制逻辑的随机结果来得到。执行时限分析常用来决定软件性能是否满足既定需求，是否能突出现有功能架构中的行为缺陷。权衡分析可以用来减轻软件性能障碍，并且解决导致死锁或颠簸状况的资源争用问题。

10.4.4　资源利用率概述

资源利用率概述描述了实际占用的资源与组件可用的总时间的百分比。资源可用性在峰 183
值负载情况下对软件产品的性能有显著影响。如果资源过度使用并且影响事务处理，那么资源的可用性可以评估为受限的。这些结果可以用来修改软件功能架构或修改计算环境定义来提高资源可用性。

10.4.5　功能规约

规约必须为每个功能组件和单元而开发。这些规约提供了功能层次中元素到既定软件需求中的可追踪性。在软件设计综合时，可以利用功能单元规约来建立结构化单元规约（见第 13 章）。工程的基本原则是对产品设计中包含的每一部分或组件进行规约。功能规约对支持软件重用也十分必要。可重用软件组件的选择是基于一个商业可用的（商用货架产品（commercial off the shelf，COTS））或之前开发的（非开发项（nondeve lopmental item，NDI））软件组件在多大程度上能满足必需的功能和性能。

10.4.6　需求分配表

需求分配表（requirement allocation sheet，RAS）可以识别那些有助于软件需求实现的软件架构和计算环境的元素。RAS 提供了关于软件功能加上计算环境特征满足既定软件需求的方式的详细信息。它提供了可追踪性，有助于研究结果和派生分配的设计决策备忘录。RAS 必须与需求可追踪性矩阵对齐，这在每个软件需求的软件工程设计工件间提供了额外的可追踪性。

184

功能分析与分配实践

功能分析与分配实践将软件需求转换为功能架构。功能分析是研究客户问题域并导出设计方案的第一步。指派给软件产品的操作活动代表了一系列复杂功能，它们需要进行分解以实施后续的结构设计方案。功能分析技术包括以下的基本任务：

1）复杂性分析——评估某个功能以确定是否可以通过将其分解为子功能进行简化。功能分解可以识别出能较清楚地看出其设计方案的更小巧、更简单的功能。功能分解的结果是一个展示复杂功能的简化、澄清和规约方式的功能层次。

2）行为分析——包含一个用来设定软件产品如何执行功能序列以完成某个操作性事务或数据处理工作的集成化建模方法。行为分析可产生包含功能流、控制流、数据流和资源使用的集成化行为模型。

3）性能分配——将软件性能需求分配到行为模型和计算环境的功能元素。软件产品性能依赖于计算环境性能特征，通过功能分析活动可对其进行提升。

4）架构评估——对演化中的功能架构进行审视以确保软件需求得到满足，设定的软件性能可承受预期的使用负载，功能架构保持简单并有助于产品生命周期中的维护和改善活动。

功能行为模型表示为获得软件产品功能和性能质量对业务模型的一种分解。导出行为模型是为了显式说明软件对各种激励和输入的反应。研究软件行为的时候，所说明的软件需求应能反映出给定计算环境限制下能够达到的性能层次。应先对不同行为可选方案进行研究后再确定进一步设计综合的方案。为了选择生命周期成本、风险和性能方面的优化方案，应通过权衡分析和风险评估对可选方案进行考量。

功能分析识别数据处理操作或者预期软件产品为支持该业务或商业过程所要做的事。功能分析不应设定结构设计或实现细节。这意味着不必完整地说明计算构造、数据结构或资源特征。软件设计综合实践为物理架构元素建立结构组成、方案和设计规约。物理架构指明了如何组织软件产品以提供指定功能，软件设计综合可对其进行配置。

本章所描述的功能分析和配置方法是从系统工程学科借鉴演化而来的。为处理独立于实现语言的软件产品设计相关的独特问题对它进行了一些改造。但是，主要的软件实现人员仍应参与功能分析活动以确保功能架构在给定的实现语言约束下能够实现。图 11-1 展示了与软件功能分析和分配相关的任务。

11.1 评估功能复杂性

功能分析与分配实施于功能架构的每个层次。形成功能层次顶层的功能初始集合是从特定的软件需求或业务模型中导出的。这些功能应代表软件产品必须执行的主要功能。围绕软件操作概念组织这些功能也许是可取的，这些概念包括某种形式的初步用户识别和认证或软件应用初始化以及主图形用户界面。在这个层次，每个功能都应被认作是复杂的，因为它们

处于业务过程或操作抽象层次。

图 11-1　功能分析与分配任务

必须对每个功能的复杂性进行评估，以确定其是否需要进一步分解来提供较简单清楚的功能描述。下列条件中任何一个成立时可识别功能复杂性：

- 功能的行为需要进一步澄清以描述从输入到预期输出或响应的转换。
- 能概念化多个提供功能的方法，需要确定一个最好方法。
- 功能涉及含混的业务规则或控制逻辑。
- 功能涉及对多个数据存储或检索事务的访问。
- 功能涉及多个来源的输入，如用户、外部系统或其他功能。

需要进一步分解的功能称为功能组件。其余的功能分析和分配任务用来：

- 定义功能组件的行为。
- 将软件性能特征分配至功能组件和单元。
- 评估功能架构的适合性和完整性。
- 说明每个功能架构元素的设计需求。

一般说来，功能分析应该持续进行直到认为其实现比较简单并且其行为无须进一步研究就能够确定。这些较简单的功能元素称为功能单元，代表配置物理架构的基本功能组件。因为与功能问题及其解决方案相关的复杂性是不均衡的，在功能体系中分解的层次数是变化的。不必勉强对层次进行统一。其目的是确保每个功能的行为得到充分理解，每个功能单元能够被明确说明和实现。功能分解可使用下列指导方针：

1）功能不能分解为单一功能。分解应确保至少识别两个子功能。

186
/
188

2）进一步分解需求由功能复杂性驱动，因此功能层次体系不必水平对称或竖直深度相同。

3）如果功能的设计方案已经能够理解或者是重用软件模块，那么就无需进行进一步分解了。

4）如果设计方案是可管理的，那么涉及多个数据转换步骤的功能也无需进一步分解。

5）如果功能涉及与其他功能相同的数据转换步骤，那么功能分解代表了建立共同子过程从而通过消除冗余降低软件复杂度的机会。

6）避免定义产生多个输出或产品的功能。通过某种考虑，可能定义一个抽象功能，然后将其分解为产生候选输出或产品的子功能。

11.2　行为分析

行为分析包括一系列用来全面研究功能解决方案空间的任务。目标是描述软件产品应该如何对用户输入、计算环境状态变更或失效条件进行响应。行为分析任务构造数据处理事务的功能设计模型。其意图是对每个事务进行的方式，包括所有可能的错误条件或计算环境失效或操作的降级模式，进行显式设计。

行为模型展示复杂功能的执行方式。应该充分详细地说明软件产品的行为以清楚表达每个功能是如何执行以便完成操作过程的。行为建模技术能够产生静态（非可执行）或动态（可执行）模型。因为动态模型能够提供对设计充分性的分析验证并能用于支持权衡分析，所以是更好的选择。因此行为模型必须包含下列元素：

1）*组织或参与者*——与软件产品交互的业务或操作组织或人员角色。

2）*外部系统*——与软件产品交互的计算环境、外部系统和其他软件应用的元素。

3）*数据存储*——支持数据存储和检索事务的抽象数据存储库。

4）*功能序列*——数据处理动作的序列流。可使用行为的并发线程来对功能序列的并发性或条件执行（基于业务或控制规则的执行流选择）进行建模（参见任务 11.2.2）。

5）*数据项*——表示功能输入和输出的信息元素（参见任务 11.2.3）。

189

6）*控制机制*——实施业务或控制规则以确定执行哪个条件分支。循环提供了对功能序列片断重复执行一次或更多次的控制机制。（注：循环可以是持续的（永无止境）或基于条件准则满足可终止的（如 Do…Until)(参见任务 11.2.4）。）

7）*计算资源*——计算环境提供的消耗性或可再用资产，如内存、数据存储以及体系接口带宽（参见任务 11.2.6）。

11.2.1　识别功能场景

软件产品支持许多应在软件需求规约中说明的业务场景。功能场景表示数据处理行为的抽象序列，这些行为对软件产品能够有助于业务或操作过程是必须的。应对每个操作场景进行评估从而识别需要加以设计的各种功能场景。例如，ATM 支持一些独特的银行（业务）交易。必须对每个银行交易进行评估来识别会触发特定功能场景的不同情况。这些情况包括：1）客户关闭导致银行账户处于非活动状态；2）不恰当的账户活动导致银行账户暂停；3）银行账户资金不足以支持所请求的事务；4）银行账户资金足以支持所请求的事务。每种情况表示必须对软件产品进行设计来支持的一个功能场景。

11.2.2　识别功能序列

每个功能场景必须按照完成场景所需的功能动作序列进行表述。功能序列可用功能流块图（functional flow block diagram, FFBD）来表示。如果进一步的条件满足多个可选功能解决方案的标识，那么序列应分支为多个序列。必须说明每个可选分支的控制规则和准则。

FFBD 表示法是一种描述功能动作序列的有效技术。但是，可能需要对其进行加强以便表达功能序列中的控制规则和可选分支。需要将控制决策逻辑和分支选择准则作为功能流序列图的整体描述来进行表达。图 11-2 展示了一个简单的 FFBD 示例，它描述图 8-3 所示的
190 处理应用业务活动的分解。

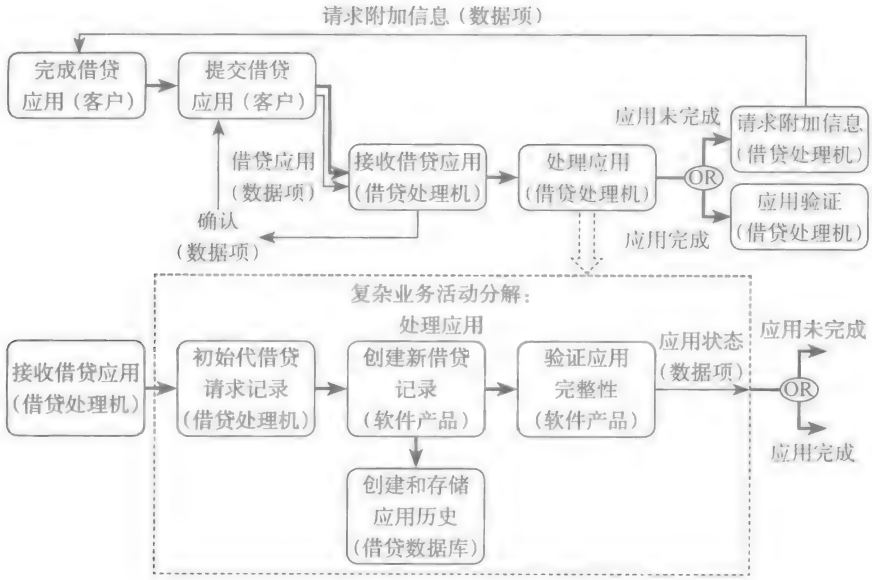


图 11-2　FFBD 示例

11.2.3　识别数据流

大多软件功能的目标是将输入转换为输出或产品。但是，有些功能接收的是控制流而非输入。只接收控制流的功能举例如下 1）显示图形用户信息屏、消息和对话屏的动作；2）对全局数据值采取行动；以及 3）当状态变量（如计算环境健康指征或软件过程状态）改变或需要进行评估（如获取默认打印机状态）时采取行动。尽管不直接处理数据的功能可能不满足与计算机语言相关的或数学的准则，但它们仍然执行一些在软件工程研究领域很重要的动作。

数据项表示功能间流动的信息。软件功能间传递的数据表示功能接口，如子过程调用。控制功能将变量传递给另一个功能对它进行处理，结果返回给控制功能。例如，一个功能可能通过传递一个华氏温度值调用一个温度转换功能。接收功能将华氏温度值转换为摄氏温度值并将
191 结果返回给控制功能。功能接口表示实现封装（面向对象中信息隐藏的一个方法）的机制。

软件功能与外部应用或系统间的数据传递表示由软件接口规约说明的软件接口。因此，功能架构必须识别这些处于功能架构边界之外的外部应用或系统以考虑软件接口。软件接口表示必须遵从的多方契约，属于接口设计和实现特征定义。

全局数据项表示所有功能都能访问的数据。因为参数值的全局可用性，这种数据项不表示数据流。软件架构中的所有功能都可使用全局数据值。本地数据项表示在功能内部说明且其他功能不能使用的变量或常量参数。因此，对本地数据项的访问不表示数据流。

数据流图用于表示软件数据处理事务的以数据为中心的视图。它将数据项表示为关注的主要元素，而将功能表示为数据项状态的连接器或转换代理。行为模型结合了功能流序列和数据流，以提供软件数据处理场景的一个更全面和完整的表示。必须建立软件数据字典（术语表文档）以确保每个数据项被唯一识别且软件开发团队的所有成员都可获得其特征。

11.2.4　识别控制行为

为了揭示决定数据处理控制流的决策逻辑和准则，必须识别控制行为。控制行为必须考虑功能场景中可能遇到的所有条件。控制行为决定在所有可能的动作路径中选择哪个功能序列。例如，一个银行职员在评估一个借贷应用时，借贷可能被批准、被拒绝或被推迟直到给出更多信息。其操作或业务模型应该识别指导该交易的业务规则。控制行为必须确定软件如何询问数据项以实施业务规则。

控制行为表示必须详细捕获的决策逻辑，以确保软件开发团队能够理解影响数据处理控制逻辑选择方式和决定选取哪个功能执行分支的准则的准确本质。用来描述控制行为的典型控制逻辑结构如下：

AND——用来描述并发执行两个或两个以上的数据处理功能行为（分支或过程）序列的并列条件。

192

OR——用来描述只执行一个功能行为序列的选择条件。执行的逻辑路径由数据项或资源的状态以及条件选择准则决定。

LOOP——用来重复一个功能行为序列直至终止重复行为的条件满足，然后执行 LOOP 后继的功能序列。

LOOP EXIT——用来评估终结 LOOP 行为的条件是否满足。

ITERATE——与 LOOP 类似，用来对功能行为序列重复指定的次数。当达到迭代的指定次数时，循环行为终止，然后执行 ITERATE 后续的功能。

11.2.5　识别数据处理过程

应通过识别将输入转换为预期输出、响应消息或状态信息所需的数据转换过程来对每个功能进行说明。响应信息与数据处理中遇到的情况认知方面的问题有关，如数据记录未找到或输入数据超出范围。状态信息与表示交易或计算环境资产状态的内部变量有关。数据处理过程应说明产生预期输出所用的算法或计算逻辑。

仅仅口头描述所执行的数据处理过程是不够的。数据处理过程应定义执行功能所需完成的转换步骤序列。例如，ATM 个人身份识别码（PIN）验证功能必须包含下列转换步骤：

1）格式化 PIN 验证请求消息。

2）向银行账户管理中心系统发送 PIN 验证请求消息。

3）从银行账户管理中心系统接收验证响应。

4）确定 PIN 验证是否成功。

5）通过下列方式告知顾客 PIN 验证结果：

- 显示顾客欢迎屏。

- 显示错误 PIN 告知屏，并请求顾客重新输入 PIN 或取消本次事务。

要判断是否需要进一步的功能分解，应重新评估数据处理过程的复杂性（见任务 11.1）。
193 功能复杂性评估的问题在于判断数据处理过程是否是一个能被合格软件业者很好设计、实现和测试的比较简单直接的动作。

11.2.6　识别资源先决条件

必须识别执行每个功能所需的资源。资源是使得功能得以顺利执行的事物。如果资源不可用，那么数据处理事务就必须暂停、延迟直至资源可用或低效执行。资源可以是功能为达成其目标所需获得的计算资源或中间数据项。

为了支持行为分析，有两种类型的资源是必须关注的。第一种类型的资源是消耗性资源。消耗性资源表示一种货物储备，它可逐渐使用直至库存清空或数量不足以支持进一步的数据处理动作。需要消耗性资源的功能在它能够执行前必须等待直到库存再次充足。消耗性资源的一个例子是配备纸张的打印机。只要纸盒中有纸，打印功能就能够执行。但是，如果纸盒中没有纸了，打印功能就会暂停直至纸盒中重新装上纸。

第二种类型的资源是可再用资源，如内存。每个功能在执行时被加载入内存，当功能不再需要驻留内存时，就被从内存中卸载。内存的数量是固定的，并暂时减去驻留功能所消耗的数量。当从内存中移除一个功能时，它为其他需要内存来执行的功能腾出了空间。及时了解任意时刻的可用内存数目对软件性能是非常重要的，它由计算机的内存管理系统规划。

资源使用是软件整体性能至关重要的一个方面。与资源管理相关的时间以及资源受限导致的延迟可能对处理时间产生负面影响。因此，资源分配和管理将对能够满足严格性能需求的功能架构的建立做出直接贡献。因此，任务 11.3.2 包括了对功能资源可用性及使用和管理系统的评估，建立了将软件性能需求分配给功能架构元素的方法。

11.2.7　识别失效条件

必须对每个功能性事务进行评估以识别可能导致失效条件的状况。识别出的失效状态必须通过确定判断失效状态所需询问的数据完整性准则和当特定状态出现时为完成数据处理事
194 务所必须采取的动作来进行处理。

某些失效条件可能导致一个不能进行自动化处理而需要人工干预的状态。那么如果可能，功能分析工作就必须阐明软件继续以降级模式运作的方式。例如，如果 ATM 的货币供应被耗尽，那么取款功能就必须暂停直到货币供应重新恢复。功能架构必须包含检测失效条件和以降级方式运作的软件功能。这包括识别软件产品被“告知”数据处理或系统资源的当前状态的方式，以及管理状态指示器的方式。

为了确定软件产品对每个失效条件应该采取的响应行动，必须对潜在的数据处理失效模式和影响进行分析。失效模式和影响分析（failure modes and effects analysis, FMEA）是一个帮助设计团队为提升软件产品质量和可依赖性对潜在失效模式基于其失效的严重性（结果）和可能性进行分类的工程化过程。可依赖性是一个比可靠性更适合软件产品的术语。在工程领域，可靠性关注在正常操作下，对硬件平均无故障时间（mean time between failure, MTBF）的预测并提供对组件预期持续运行时间期望值的预估。软件不随使用时间而磨损或毁坏。而可依赖性则指软件组件在所有状况下执行预期功能的能力。因为组成软件的原材料的本质，可依赖性是一个更适用于软件产品的术语。如果一个软件组件失效，原因往往是其

软件设计不具备处理非预期状况或条件的弹性。

应该应用软件 FMEA 来识别潜在失效模式，确定其对系统操作或业务过程的效应并设计避免失效发生或减少失效对操作性能影响的响应机制。尽管不可能预料到所有的失效模式，开发团队应该以下列方式制定一个潜在失效模式的丰富列表：

1）开发最小化潜在操作失效可能性的软件产品需求。

2）评估涉众在软件性能和开发后的过程方面的需求，以确保这些需求没有引入复杂的失效条件。

3）识别有助于失效检测和减少失效在数据处理事务中传播的设计特征。

4）开发用以演练与失效检测、隔离和恢复相关的软件行为的软件测试场景和过程。

5）识别、追踪和管理潜在设计风险，以确保产品可依赖性可通过软件测试活动进行预测和证实。

6）确保可能发生的任何失效都不会导致人身伤害或严重影响系统或操作过程的运作。 [195]

正确使用软件 FMEA 可为开发团队提供很多益处，包括：

1）提高软件的可依赖性和质量。

2）提高客户和涉众的满意度。

3）在软件开发过程早期，这些设计问题可被事半功倍地进行处理时，识别和排除潜在的软件失效模式。

4）强化的失效检测和避免措施。

5）为提高软件测试覆盖率提供一个关注点。

6）减少后期的设计变更及其相关的成本和进度影响。

7）提高开发团队成员间的团队合作和交流。

对软件产品的完整 FMEA 应该处理在计算环境硬件、外部系统和数据处理事务中发生的失效及其对最终系统或操作过程的影响。软件 FMEA 过程应遵循依据 IEC 60812 改进的以下步骤[⊖]：

1）定义分析的软件边界（在计算需求分析中获得）。

2）了解软件需求、功能和性能。

3）建立功能架构表示（层次分解和行为视图）。

4）识别功能失效模式并归纳失效影响。

5）建立失效检测、隔离和恢复的准则。

6）报告结果。

11.2.8　识别系统监控过程

如果软件产品涉及对机器或其他类型设备的控制或监控，那么软件 FMEA 应该识别必须处理的状况。必须对系统监控、状态通知和修正行为功能进行识别。应在行为模式和功能体系层次中对系统监控和控制行为进行强调，这样它们才能在软件实现和测试中得到重视。因为系统监控和控制行为功能的重要性，在软件架构进行演化时，应该彻底评估对系统监控和控制行为的变更。

必须识别系统状态并说明软件监控功能的周期。必须将与软件响应系统状态变更相关的

⊖ 系统可靠性分析技术——失效模式和影响分析过程（FMEA），国际电子技术委员会，2006.1.25.

行为整合到功能架构中去。应对系统健康状态检测进行实时或准实时监控，并为降级系统运
196 作条件定义软件响应。必须为安全攸关系统的状态变更建立模型或原型，以确保软件能够正确地检测和执行维护系统运作条件所需的修正动作。

11.2.9　识别数据保留能力需求

必须对用于长期数据保留记录的数据存储能力需求进行说明。应该对操作或业务模型进行评估，以确定一个给定时间段内所必须支持的预期最大数量的数据记录。应该使用操作预测来判断数据存储能力在不同时段的需求。在进行能力规划时必须考虑的因素有数据存储设施的位置、数据记录保留时间、被删除数据记录存储空间的恢复和不同时段的新数据记录创建需求。数据保留能力需求不仅将直接影响计算环境的配置，还将影响到软件与数据管理系统的交互。

11.2.10　识别数据安全过程

必须识别保护机密或保密信息的数据安全功能和过程。信息安全是一个研究更大范围的计算机安全和信息保密问题的专业领域。数据安全是软件产品将要具备的信息安全能力的子集。信息安全意味着保护信息和信息系统免于非法访问、使用、泄密、破坏、修改、查看、检查、记录或毁损。软件工程应涵盖对敏感信息（机密或保密的）的访问进行监控和管理的逻辑控制的建立。必须针对以下方面识别信息安全功能并定义合适的过程：

- *访问控制*，包括用户账户管理、识别、鉴定和授权。访问控制通过限制被授权访问敏感信息的人员的方法来保护信息。
- *信息安全分类*，涉及对不同数据保密级别、赋予数据特定级别的准则和管理各级别敏感数据访问所需的控制的识别。
- *密码体系*，包括信息加密与解密。

11.2.11　识别数据持久性与保留功能

必须对数据元素进行评估，以识别临时事务持久性和长期保留的需求。如果需要获取事务状态以支持撤销或逆向事务步骤来返回到早前条件的操作状态，那么就需要临时数据持久性。如果必须保留历史数据以支持事务记录保留、统计分析或其他业务功能，那么就需要长
197 期保留数据。必须对数据持久性和保留需求进行说明，以支持软件操作所需的数据保留机制的定义。

附加功能应该整合到包含数据持久性的行为模型和功能层次中去。数据存储和检索事务应该需要处理数据库可靠性问题、数据验证过程或可能的事务死锁状况。随着功能架构逐渐成熟，数据存储和检索功能应该进行组织、说明并记录在数据库事务文档中。为支持操作或业务数据保留需求，数据库事务块图应识别设计数据库所需的数据记录、数据类型和定义。

11.3　性能分配

必须将软件性能需求分配给功能架构的元素。这种分配必须考虑每个软件功能的持续时间以及与数学运算相关的精度和准确性。软件功能性能应该只考虑与完成软件任务相关的执行时间。确定运作或业务过程的性能时应该考虑与用户输入输出以及同外部系统和其他软件产品相关的性能。

软件性能依赖于计算技术的执行和充足硬件资源的可用性。如果对共享资源的访问变得繁忙，延迟了功能执行或延长了功能的执行时间，性能瓶颈就可能出现。

瓶颈是整个系统的性能或能力被一个或少数组件或资源所限制的现象。术语瓶颈来自“资产如水”的比喻。当从瓶子里往外倒水时，水流出的速率受限于出口通道的宽度即瓶颈。通过增加瓶颈的宽度，可以增加水流出瓶颈的速度。因此系统的限制组件有时被称为瓶颈。⊖

因此，必须将软件性能需求分配给功能架构元素。这不仅涉及软件执行时间，而且涉及合理的资源使用以避免资源导致的性能降低。 198

11.3.1　分配性能预算

必须对每个功能的执行性能进行说明并将之分配至组成它的子功能。执行性能与软件功能在特定计算环境中执行所需的时间有关。软件功能的持续时间不应包含与外部用户或系统交互有关的任何延迟。性能需求的初步分配应该认为是预算性的，直到软件功能架构的设计完成。然后产生的功能规约必须说明与每个软件功能相关的性能需求。既然功能架构会不断演化，性能预算可能会随着软件实现专家对可完成和交付系统预期的改进会不断发生变化。

功能所需时间可以说明为一个不变或可变的时间区段。当功能所需时间能保持相对一致时，应该使用不变的时间区段。可变时间区段定义应使用概率分布函数来表示功能的随机执行时间。

行为模型分析应该对功能执行的关键路径进行深入考察。关键路径分析是识别高度并发系统中瓶颈的一个强有力的方法，但是它通常需要丰富的领域知识来构造所需的用来识别软件行为模型中事件之间依赖和时序关系的事件图。具体功能的关键程度可以定义为该功能持续时间与关键路径总时间的比值。该度量给出了对数据处理系列中最重要功能的一个快速归纳并为关键路径提供帮助。该分析可以识别功能架构中能够为提高整体软件性能提供最大机会的区域。

11.3.2　分配资源预算

为了在软件功能中平衡资源利用，应该进行行为模型分析。当数据处理功能并发执行并需要对有限资源对象进行访问或控制时，资源就成为一个约束。资源对一个功能不可用时响应为拒绝服务，这将使该功能进入等待状态（即等待资源可用）。某些资源可通过对请求服务的功能需求进行排队或组织来进行管理。资源请求优先级为管理资源和确保以最高效方式完成数据处理动作提供了基础。

初步资源分配应该认为是预算性的，它将随着软件设计和实现的完成进行调整。性能预算表示每个物理架构元素的预期目标。结构元素规约应识别实现的性能需求，但是，这些规约可进行调整来反映实现中达到的实际性能。为了建立性能特征并帮助资源利用策略优化， 199
应该进行相关软件实现和测试活动。

11.4　架构评估

必须不断对形成中的功能架构进行评估以确保它能够满足软件规约并且不会过度复杂以

⊖ 参见 *http://en.wikipedia.org/wiki/Bottleneck*.

致影响到软件维护成本。这些架构评估任务为软件工程团队提供了对功能架构能作为软件产品生命周期支持的有效基础并将有助于未来加强和扩展的再保证。

11.4.1　评估需求满足

当功能架构逐步演进时，必须持续对它进行评估以确保它与软件相符。功能规约集合应该能够追踪到软件需求规约和涉众需求与预期。演化中的功能架构的充分性主要通过确保软件功能分解保持相对简单以及软件需求规约能被实现来进行判断。

11.4.2　评估软件性能

必须对软件行为进行评估和调整以确保性能需求被满足。功能所需时间和资源使用策略应该进行同步以对数据处理请求提供适当的软件响应。

11.4.3　评估架构复杂性

必须对功能架构进行评估以确保行为复杂性不会对未来的软件提高产生负面的阻碍。软件行为必须有效满足操作或业务过程。

11.4.4　评估优化机会

必须对功能架构进行评估以识别改进的机会。优化功能设计（分解和行为）的成本必须通过性能的提升或设计复杂性的降低得到回报。

11.5　建立功能架构

200

为了建立软件设计综合的功能设计基线，功能架构必须置于技术配置控制之下。功能架构必须是完整的，并能够追踪到软件规约。软件工程团队中的软件实现、测试和评估组织成员必须支持功能架构并修订其技术方案和进度安排以使得组织资源与预期任务分配相匹配。

必须为功能架构建立文档以提供相关的图解、图样、模型和规约，软件设计合成可依据它们进行执行和评估。功能架构包括表 11-1 所示的设计文档。

表 11-1　功能架构设计文档

文档标题	文档描述
1. 功能分解描述	该文档描述软件功能分解为子功能的方式。功能层次体系图展示功能需求分解为组件和单元的层次
2. 功能组件规约	对软件功能配置的复合元素的软件功能和性能规约。这些规约表示支持更高层功能组件或软件产品需求实现所需的子需求
3. 功能单元规约	软件功能配置基本元素的软件功能、性能和设计规约。这些规约表示支持底层功能组件实现所需的子需求
4. 功能接口规约	每个软件功能接口的技术描述。识别接口的目标并提供与通过接口所交换信息的类型相关的一般信息
5. 软件行为模型	行为模型描述复杂功能的功能序列、控制和数据流。行为模型为每个行为线程建立软件性能和资源使用规约。行为模型描述失效模式和效应，还应该描述错误检测、隔离和恢复过程
6. 数据持久性规约	记录数据持久性需求，包括数据存储能力需求和数据存储和检索事务
7. 需求可追踪性矩阵	记录软件规约中的每个需求如何被功能架构中的元素所满足
8. 软件术语文档	记录功能架构中定义的每个功能、数据项和资源的特征以提供命名元素的统一术语表

201
/
202

物理架构配置

本章讨论如何从功能架构中导出并配置软件的物理架构。通过功能的分析和分配来进行软件功能解决方案的分析、说明和简化。软件设计综合过程中，建立了产品的各个结构要素以及这些元素之间的排列、组合和集成关系。软件设计综合的结果是软件产品的结构配置，这与它的文档和物理架构相一致。完整的软件设计方案包括功能架构和物理架构。软件产品的架构是由软件产品的规格、功能、物理架构以及计算环境架构所组成。完整的软件架构是软件产品的架构以及软件开发后的过程的架构。图 12-1 显示了不同的架构配置之间的关系。

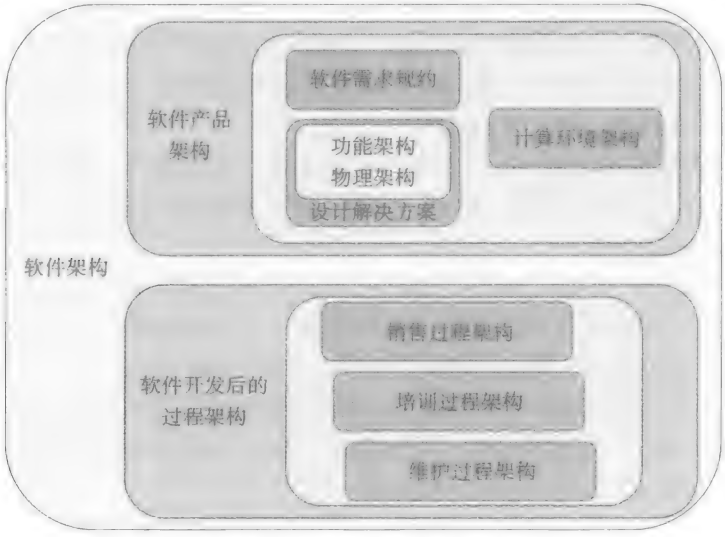


图 12-1　软件架构配置关系

物理架构通过收集设计方案的结构表征、工程图纸、图表和模型来设计方案，它规定了基本构建块（结构单元），从而该软件产品能够依据其进行实现。软件实现的初步实施阶段中，这些结构单元规约交汇出软件单元的设计、编码和测试方式。本章将进一步讨论如何在软件设计综合过程中对结构单元进行识别和规约。软件集成策略解决了软件组合、集成和测试的方案，从而产生完全集成的产品配置。最后讨论如何使用建模与仿真方法来帮助设计方案决策。

物理结构是由技术图纸、图表、规约和定义结构设计解决方案的模型文件组成。结构设计解决方案规定了结构单元和组件的配置，以及它们是如何组装和集成到一个单一的产品之中。结构单元和组件的规约为软件的实现提供了基础，其中涉及软件单元的设计、编码、测试，以及软件组件的集成和测试。因此，物理架构导致了软件产品的技术数据包，为软件的实现提供了必要的结构设计文档。技术数据包的内容为后文中所论述的“准备软件技术数据

包”部分。

软件工程面临的最大挑战是确定如何将功能架构变换到物理架构。功能架构建立基本的数据处理任务、数据流和逻辑控制，以及软件设计方案的资源分配；而物理架构是软件结构设计方案中的软件模块或结构单元，以及这些单元是如何相互作用和集成为软件组件。因此，本章讨论了开发物理架构的以下挑战：

1）如何识别和规约结构单元？

2）如何设计软件集成策略？

3）如何简化结构设计配置来降低复杂性和维护成本？

4）结构设计解决方案如何转化成软件实现？

12.1　结构设计解决方案

结构设计解决方案为软件产品配置项建立了结构单元和组件的组织和集成关系。一个软件产品可能由一个或多个配置项构成。客户端/服务器软件产品将包括客户端配置项和服务器配置项。结构设计解决方案还确定了软件产品配置项之间的接口和计算环境中、外部系统和其他应用软件的元素，如数据库管理系统（DBMS）。软件产品概念设计方案的框图如图12-2所示。这个框图显示软件产品配置的元素之间的集成关系，以及软件产品和外部实体之间的接口关系。图12-2中的元素包括：

1）*软件结构设计解决方案*——确定软件产品配置、结构组合、软件集成策略和特征化软件设计的接口。设计工件（图纸、图表、模型、规范和文档）描述了产品配置，其中包括软件产品技术数据包（TDP）。

- *软件产品配置*——明确结构单元和组件，以组装和集成为软件产品。
- *结构单元*——代表基本设计元素或软件模块，用于组装软件的结构组件。用来表示一个结构单元的典型术语包括模块、程序、过程、函数和对象。
- *软件组件*——代表集成的软件模块，由两个或两个以上的结构单元或底层组件构成。

205

2）*软件集成策略*——明确将更大、更复杂的结构组件组合为最终软件产品配置的组装和集成任务序列的顺序。

- *结构组合*——代表组装和集成的任务，以生成工程测试和评估所需的测试桩和结构组件。
- *图形用户界面组合（例子）*——结构组合的一个常见实例为基于图形显示的屏幕、菜单、图标以及指向设备接口的结构单元和组件。

3）*软件运行环境*——标识计算设备、外部系统，以及相关的软件应用程序，它们与软件产品的运行相兼容，能够协作完成任务。

- *计算环境*——软件产品用于协作的计算技术资源集合。
- *接口关系*——写明软件产品的协议和承诺，以与其他系统、设备项目或软件应用程序交换数据。一个接口可能涉及单个来源、单个目标、单个方向的数据流信息；也可能涉及多重来源、目的地、双向的数据流信息；或者它可能是一个广播类型的接口，以传输来源广泛的数据到所有合格的接收者。
- *外部接口*——写明数据的交换协议和格式，使得数据能够共享于接口（交流）系统和应用程序之间。一个接口是一个系统开发人员之间的正式协议，符合双方的接口控制文档（该文档描述了接口）。

a. 外部系统——正在开发的软件产品必须与外部系统或软件应用程序协作，实现接口的目标。

b. 数据库管理系统（例子）——外部软件应用程序的一个常见实例是软件产品可能被设计成操作和界面。

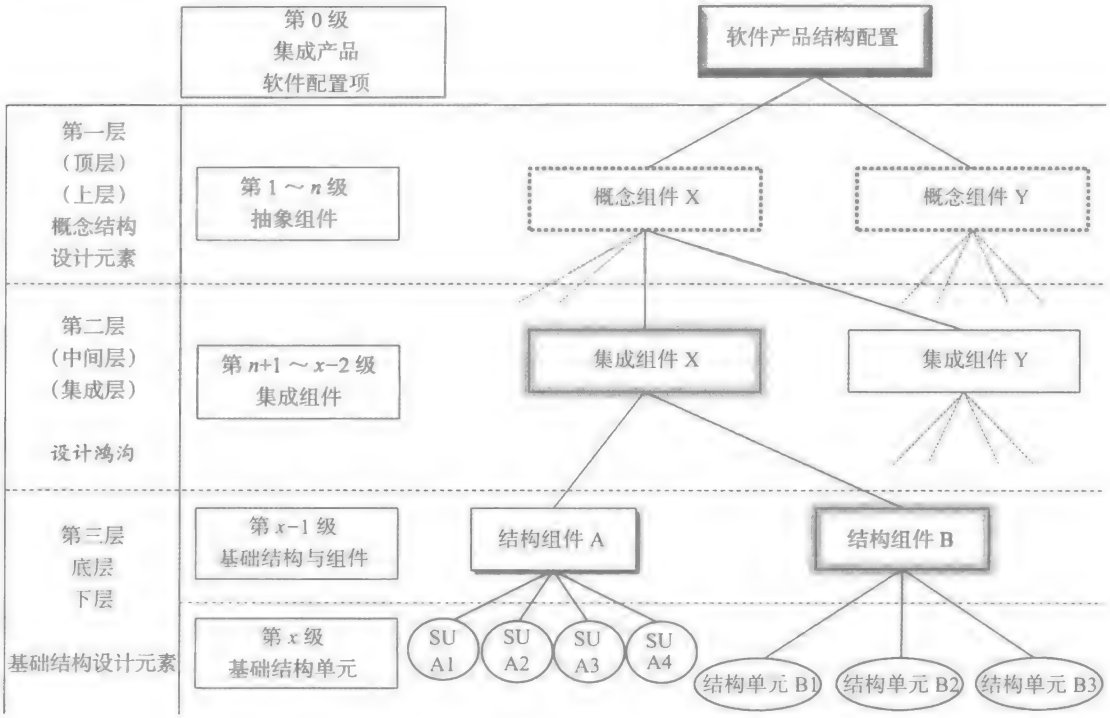


图 12-2　软件结构设计方案

软件产品配置包括诸多结构单 元，这些单元通过组合和集成形成结构组件。结构组件组成更大的配置元素，包括一组集成的结构单元和 / 或组件。集成的结构组件用来描述软件产品配置，旨在向客户、利益相关者或消费者交付。结构单元和组件是软件产品的组成元素，均需在整个软件生命周期中进行实现、测试和管理等活动。

同时，可能还需要创建组合配置（结构组合）来支持软件的集成和测试。结构组合代表了一个中间的工程结构，包括额外的软件（测试桩），以支持软件集成和测试（如测试工具等）。

结构组合定义了结构单元、组件和测试桩，它们通过组装和集成来支持软件测试。结构单元没有出现在结构组合配置之中，因为它们在结构组件组装和集成的第一级别中就已经集成。支持结构单元测试和评估的测试桩与结构组合配置无关，因为它们在结构组合层次的集成和测试中已经失去了效用。

12.1.1　定义结构单元

功能架构之中的功能单元集合定义了结构单元。结构单元并不是一个与功能单元一一对应的集合。定义结构单元的目标是建立一个最小但完整的结构单元，可以配置软件产品。结构单元最终将成为原始软件实现工作的焦点，包括对各个软件模块、程序或对象进行设计、

206
～
207

编码和测试。因此，每个结构单元的设计必须考虑功能分配，需要与其他结构元素相互作用，通过必要的努力实现集成和互操作的功能和性能。

功能架构必须通过评估，对指定的一个结构单元确定一组公用的或紧密耦合的功能单元。这并不禁止单个功能对应单个结构单元。然而，此项工程实践的挑战在于如何组织指定的功能，以实用的方式提供一个有效的和高效的解决方案。必须考虑影响设计方案随着时间发展的几个因素：

1）未来扩展或软件功能增强的潜力。

2）能够接受或适应计算环境变化的能力。

3）软件产品演化以修复设计或编码缺陷的能力。

4）组装软件产品所必需的集成和测试活动的顺序。

结构单元应该按照关键功能划分为不同类别，如用户界面、数据库事务处理、业务流程功能、用户管理、资源管理、错误处理和数据安全。结构单元识别的准则包括：

- 一个功能单元是独立的动作执行，没有类似的功能单元与它组合。
- 一个功能单元在指定与之对应的结构单元时是有问题的、有风险的或需要特殊工程注意的。
- 一个功能单元需要大量的建模、仿真或测试工作来保证与之对应的专门原型和结构单元。
- 多个函数执行类似的数据处理操作，可以将其结合，但是不能增加结构复杂性或超过结构单元既定大小。
- 多种函数在共同的数据集上执行数据转换时，这些函数要保证一致性且符合整体结构配置。
- 无论结构的既定大小如何，类似功能单元的结合都会加强结构设计方案的复杂性。
- 类似的功能单元的结合将对软件组件集成和测试带来不利影响。
- 类似的功能单元的结合将影响开发后的过程中相关的修改或规划。

208

软件项目物料清单（BOM）中会对结构单元逐项列举。[⊖]软件 BOM 建立结构组件和单元的层次结构，形成软件产品配置。它用于支持软件的构建过程，将源代码转换成可执行代码，按正确顺序连接在一起，直到生成一套完整的可执行文件。软件构建过程利用 BOM，提取软件源码文件之间的依赖关系知识，链接可执行文件，形成一个完整的软件产品配置。

12.1.2　准备结构单元规约

每个结构单元必须与软件实现明确对应。结构单元按如下方式获取其规约：对应多项功能单元的结构单元，结合和吸收相应功能单元的规约，消除重复，解决相互冲突的需求，达到总体性能参数的要求；对应单项功能单元的结构单元，继承需求规约中包含的该功能单元的规约。在这两种情况下，结构单元规约还必须处理结构需求，其中包括：

- 源代码行（软件大小）的估计。
- 内存利用率和资源利用率的估计。

⊖ 参见 http://en.wikipedia.org/wiki/Bill_of_materials。清单中是一系列的原材料、配件、中间件、子组件、组件、部件，以及开发一个最终产品所需各项的数量。BOM 可以定义产品的设计过程（工程 BOM）、或开发过程（制造 BOM）、或维护过程（服务 BOM）。BOM 的不同类型取决于业务需求和使用方式。BOM 是天然分层的，而其顶层则代表了最终产品。描述组件的 BOM 称为模块 BOM。

- 内部接口定义（软件之间的接口）。
- 外部结构定义。
- 计算环境交互和诊断。
- 实时执行约束。
- 同步数据处理执行和调度。

12.1.3　建立软件集成策略

结构组件的初始层次应根据需要确定，将常见的结构单元合并成更大、更复杂的组合。在这个初始层次上，仅由结构单元组成的结构组件视为基础结构组件。每个基础结构组件的任务是组装和集成一组结构单元为一个可执行文件，能够进行集成测试。在后续的结构组件层次 209
中，通过合并子组件和 / 或单元，形成更大的组件。这些集成层次中的中心组件称为中间结构组件。将结构子组件和 / 或单元集成为中间结构组件，这一过程应该持续，直至最高层次的组件被识别出来。这将形成最终的基本结构组件集合，据此，可以组装出软件产品的配置。

每个结构组件的规约必须结合和吸收相应集成元素的规约。这些结构组件的规约不应该形成独立的子结构元素（组件或单元）的功能，而应该形成功能或事务处理，实现或提供集成工作的结果。

单个组件的集成测试策略必须确保集成组件的执行令人满意。然而，集成测试策略的重点应该是确保集成对组件需求的影响最大。如果结构组件需求不受集成的影响，那么软件开发工作可能仅仅通过了子组件或单元级的测试。

组件集成层次的数量反映了软件设计方案的整体复杂性。因此，软件集成策略的制定应遵循以下准则：

1）每个结构组件必须是按其组件集成规约独立可测的。

2）结构组件应该在结构集成层次提供一个更有能力的元素。因此，每个结构组件应该能够被追踪到一个操作或业务流程中的重要任务，或者应该满足一个或多个软件需求。

3）结构组件应该能够作为一个“黑匣子”被测试，测试的重点是验证集成的子组件和 / 或单元之间的内部接口，以及测量组件的性能。黑盒测试不评估内部逻辑、行为或集成元素的算法。它应当着眼于集成工作的结果，即集成的功能和结构组件的性能。

4）参与软件集成策略的集成任务数量应该尽可能地压缩。过度的集成任务会增加软件开发成本，延缓软件开发进度。与软件集成和测试相关的负担将在软件产品生命周期的每次更新和发布中反复出现。因此，每个集成任务应详细检查，以确保其必要性和成本效益。 210

12.1.4　指定工程组套

需要评估集成层次结构中的每个结构组件，来确定该组件是否需要额外的驱动程序或桩来测试。对所需的测试驱动程序或桩都应当进行识别并纳入软件实现规划之中。这作为与结构组件相关的一个工程组套，记录在物理架构之中，标记相关的结构组件是否需要扩大实施内容来支持软件集成和测试。该工程组套必须确定测试驱动程序和桩，必须开发相应的工程规约来表达由其增加的软件实现任务。

12.1.5　准备软件技术数据包

软件 TDP 用于在软件实现过程中为每个开发元素（包括结构单元、组件和工程组套）建

立相关的文档。软件 TDP 包括与结构单元和组件相关的技术文档（图表、图纸、规范、数据定义和软件集成策略）。软件 TDP 形式化表示了软件实现阶段所必需的物理架构。而软件的实现涉及设计、编码和结构单元的测试，以及结构组件的组装、集成和测试。保证软件物理架构的一致性、明确性和文档化，将有效改善与软件实现和测试相关的工作。

对于软件实现、验收测试和开发后期软件维护而言，TDP 就是对支持这些活动的所有软件产品所进行的技术描述。因此，TDP 必须符合软件产品架构的演化，并兼容所有授权的变更请求和提议。TDP 与软件开发文件夹、验收测试结果以及授权的变更提议、偏差和消除等内容一起，提供了软件的基础功能配置和物理配置审核（FCA / PCA）。

12.2　结构设计考量

软件产品提供了大量的各种行业的技术方案。因此，物理架构的配置依赖于设计方案的类型。本节为建立软件设计策略、过程和方法中所需要考虑的问题提供了通用指导原则或建议。

12.2.1　结构设计指导原则

211 某些软件工程原则适用于所有软件产品，而无论应用程序开发属于何种类型。软件工程过程是基于一个事实上的自顶向下的方法来建立软件功能架构。然而，配置物理架构本质上是一种自底向上的方法，在进行结构设计时必须考虑以下基本原则，以精心制作软件架构：

1）功能分析试图有条理地处理问题和方案的未知信息，因为它涉及问题的分析和解决方案空间，这意味着使用自顶向下的方法将一个大型的、难以理解的问题或情况逐步减少到更小、更一致的设计挑战。

2）功能分析试图解决复杂的任务，分解每个任务，直到一组更简单、复杂度更小的功能。

3）通过应用功能分析方法所得收获自然需要向上迭代，重新考虑提振后的问题和解决方案空间。

4）物理架构通过设计综合，力求将软件元素组合为一个新的、更大、更复杂的软件元素，有效和高效地改善整体设计方案。

5）应用设计综合的默认方法是：从最小的部分开始，按照设计方案和进度，装配结构单元和组件，将其集成到更大、结构更复杂的组件之中。然而，工程分析过程中需要识别包含最多风险的设计挑战，并寻求包含更少风险的设计方案。然后，剩余的设计可以利用风险更低的设计方案作为其核心，加以解决。

6）对于一个合适的、强大的设计方案的追求，往往会颠覆一个严格的自顶向下功能和自底向上的设计综合实践。设计的效率和有效性优先于对严格架构设计策略所指定设计方法的遵从。

7）软件工程混合了自顶向下分析和自底向上的综合技术，这是因为我们了解了结构设计必须按照一个抽象的自顶向下的方式进化。然而，在彻底应用自底向上的设计综合实践之前，抽象结构设计一直是代表性的设计方法。

软件产品不受任何科学、技术、工程或数学原理的束缚，它们为其他产品提供了设计参考模型。在今天的市场上绝大多数的产品是基于现有产品开发的，只有轻微的创新变化。许多软件产品的架构设计没有沿用传统遗留的设计来进行。可供参考的软件架构可以建立一个

域或软件产品线。但是，涉众的需求和计算机技术的进步往往展示出这些参考架构是低效或无效的。因此，大多数软件产品的设计工作从零开始。以下为如何建立软件结构设计配置提供了指导原则：

1）评估高层的功能架构，因为它提取制定软件产品结构配置的核心元素。这涉及在功能组件之间寻找共同的主题，从而识别抽象结构元素，以提供组织、监督和监控。由于所有软件产品都是进行数据处理的一种形式，我们面临的挑战是建立一个顶层结构组件的抽象的设计配置，用于产品功能的分配。组织、监督和监控结构问题需要在此解决，才能够为抽象软件设计配置的建立提供额外的指导。

- 基于组织结构的设计依据业务流程的执行方式建立了其配置。这会确定设计配置的类型，可能是面向过程的、任务导向的或是基于角色的。
- 基于监督的设计依据为用户 / 操作员与软件功能执行间的交互提供指导的需求建立了其配置。一个常见例子就是文字处理或 CAD / CAM / CAE[⊖]应用程序，它提供了一组下拉菜单来访问产品的主要功能。
- 基于监控的设计依据对过程控制系统和设备产品状态进行观察并在系统表现异常时及时采取纠正措施的需求建立其配置。主要的结构组件围绕如下项目组织：要控制的实时过程、要监视的系统或设备项，或计算机监控系统中可能发生的故障类型和所采取的纠正措施。

2）随着功能架构的演化，抽象结构设计配置可以被扩展，以包含与每个结构组件功能相关的知识改善。这涉及识别抽象子组件来对较低级别的功能进行组织。这可以与功能分析实践同时进行，来演进对于功能架构的理解。当功能架构完成时，功能单元必须分组，并作为构建模块或部分分配给结构单元，进而进行软件产品的组装。结构单元必须与抽象结构组件一起达成初始的结构设计配置。这对于不论是自顶向下方向还是自底向上方向的设计工作都是适用的，可以为目标物理架构建立一个全面的结构配置。

3）对结构配置进行评估，以改善设计来优化性能，减少软件集成和测试工作，简化配置。

4）通过识别中间结构组件，形成结构组件的逻辑集合来建立软件集成策略，组成最终的软件产品配置。这些中间结构组件是抽象结构组件的子组件。基于此项任务，有理由重组结构配置，抽象组件可以完全消除或重新审视以适应更广泛的功能职责。最终的软件集成策略应该确定软件实现过程中对结构单元和组件进行组装、集成、测试的方式。此集成策略之后，软件构建过程将被模式化而不需要再进行中间结构的集成测试。

这些软件设计的指导原则强调功能分析和设计合成以结构配置与功能架构相协调的方式进行应用。功能分析建立了软件产品的功能和性能特征。它提供了一个定义良好的、有组织的方法来探索问题空间，并在涉众需求和软件需求规约可能含糊不清的情况下，对操作特性和属性进行深入了解。

设计综合使得软件工程团队能够推断出一个概念设计的框架，而同时仍需阐述功能架构。概念设计应着眼于可增加的结构组件，关注软件产品配置的框架结构。通过对设计的主导功能概念进行抽象，获取一组高层的抽象概念结构组件，形成概念设计框架。

在功能架构设计完成时，设计综合实践用于分组和组织功能单位，形成一套完整的结构

⊖ CAD/CAM/CAE 分别是计算机辅助设计、计算机辅助制造、计算机辅助工程的缩写。

单元和结构配置的构建模块。结构单元规约通过功能单元规约进行整合，形成一个综合的需求。我们需要建立软件集成策略来识别结构组套和中间结构组件，以及连接结构单元的设计空间和概念设计框架。结构组套代表中间结构组件，它包括支持组件集成和测试的额外的测试桩和驱动程序。由设计综合产生的完整的结构配置必须要完整说明和记录，以提供软件技
214 术数据包交付给软件实现团队。软件的物理架构涉及由软件 TDP 记录的完整结构设计。

12.2.2　使用建模与仿真

所有工程学科的目的是使用数学或科学原理设计、计划、构造或维护产品来为一个问题设计解决方案。产品设计几乎总是以模型的形式展现，该模型能够以比初始属性更少的属性完成产品设计特性的表达。工程图和图纸是描述制造、组装、施工、维护或工程细节的静态模型形式。仿真是动态的模型，用于支持软件行为的实验，以收集关于产品或过程设计的信息。建模与仿真的结果用于改进工程特点和设计，进而改进产品性能、可靠性[⊖]和生命周期的品质。

建模与仿真是对模型的使用，包括仿真、原型、模拟器和仿真器，或为静态或随着时间变化，以此开发一系列数据作为架构决策的基础。建模与仿真这两个词语往往交替使用。[⊖]模型是将要设计的产品或过程的静态表示，而仿真则是动态表示。

模型和仿真都是工程工具，用于：

1）将产品特性传达给涉众。

2）对具有挑战性的工程和设计特点，如性能、互操作性、用户输入可接受性和可用性，证明其是否实现。

3）依据各种各样的工程权衡，如成本效益分析、可行性、可支持性和资源利用率，评估候选设计方案。

4）为产品设计配置、接口、行为以及集成和测试程序提供工程表示（图纸、图表、可执行模型）。

5）支持培训和教育用户如何正确使用或探索该产品。

6）对过程进行定义，以支持过程分析、设计和评价。

工程模型有多种形式，并可以构建在许多材料和媒体格式上，如纸、粘土、木材、电子表格、图画、图表和基于 CAD / CAM / CAE 的计算机表示。工程仿真是动态的模型，用于
215 评估在相关操作和环境条件下的设计特性。

模型和仿真在整个软件工程实践之中可以用于支持多种任务，以表示设计概念，评估设计能力和质量因素，以及收集关于软件产品的有效性或开发后的过程设计的反馈。这种反馈能够帮助软件工程团队理解涉众的需求和期望。下面的章节主要讨论在软件工程实践中使用建模与仿真。

12.2.3　行为分析

行为的建模与仿真能够支持功能分析和设计综合实践，因其能够提供一个全面的、集

⊖ 可靠性是指产品在各种操作环境下表现恰当的可信程度。这是一个术语，更适合于基于软件的产品，相比之下，可依赖性、可用性和可维护性更适用于基于硬件的产品。

⊖ 参见 Department of Defense Modeling and Simulation (M&S) Glossary, DoD 5000.59-M,U.S. Department of Defense, 1998.

成的软件架构（产品和流程架构表示）。行为模型捕获了架构元素之间的关系，将设计元素、特征和功能回溯到涉众的需求和软件规约。行为模型应当从以下设计表示或模型的角度表示软件架构：

1）功能分解图——描述功能层次结构，捕捉分解的功能组件和单元之间的双向关系。

2）业务模型——描述操作或业务流程，将其表示为功能、数据和控制流的集成视图。业务模型是功能流程框图、数据流和控制流图的复合视图。

3）执行时间表——描述软件产品或过程的执行时间表，确定功能排序、数据交换的持续时间与资源利用的图表。业务模型应该提供一个仿真功能，可以自动生成执行时间表。

4）实体关系图——描述软件架构的单个元素及其与架构中的其他元素间所建立的关系。

5）接口模块图——描述结构配置的每个元素的物理接口，其中包括该元素与其他结构元素、外部系统或应用程序之间的配置。

6）结构配置图——描述结构元素组成的软件产品结构配置、结构组合或组件。它用于将软件产品或结构组件分解成低层结构元素（组件或单元）。（注意：在自底向上的方式中，该图还应该确定软件集成策略。）

7）工程组装图——结构配置图的一种，用于确定结构的子元素、测试桩或驱动程序，以对结构组件或软件产品配置进行必要的组装、集成和测试。 216

8）软件集成图——结构配置图的一种，标识每一个结构元素的版本、文件名称和位置等内容，以对结构组件进行组装、集成和测试。

图 12-3 为软件产品架构提供了一个行为分析设计的例子。对软件产品架构的设计表示的识别适用于开发后期每个阶段中的架构。（提示：集成产品与过程开发（简称 IPPD）理论指出，软件工程实践适用于软件产品的设计和开发后的过程。）

图 12-3　行为分析设计表示法

12.2.4　结构权衡分析

对两个或两个以上的、针对相同的操作或环境条件的设计方案进行分析评估时，任何信息收集活动都属于权衡研究。必须对候选的设计方案进行评估，以了解各种产品特点并支持

工程决策。设计权衡通常会涉及关键性能、操作和维护成本因素，这些因素会影响产品的适用性以及产品维护和项目目标的实现。

需要考虑各种各样的物理设计方案，满足功能架构的需求，才能获得一个合理的设计方案。必须对这些设计方案进行分析和评估，以确定哪些候选方案就满足涉众的需求、需要和目标而言提供了产品和项目特征的最佳权衡。

软件分析实践是软件工程的一个组成要素，用于支持与软件需求分析、功能分析与分配、设计综合等相关的权衡分析和风险评估。软件分析将在第 14 章讨论。

12.2.5　软件产品性能评估

最终，软件产品设计方案将在如何有效地执行数据处理任务这一点上进行评估。软件工程中涉及许多“对产品响应持续评估的刺激”（包括用户和外部接口），通过这些刺激来判断软件产品在各种运行情况和条件之下的可靠性，以及它利用和节约计算资源的情况。下面的章节将讨论这些关键性能的调查过程中的挑战。

1. 设计响应能力

217 软件设计的响应能力包括软件产品对用户的输入、外部接口的刺激或与计算环境的交互等内容的及时响应能力。软件结构设计必须通过评估，以保证该设计可以提高软件产品对请求操作的响应能力。以下指南对于如何提高软件对用户请求的响应能力给出了解决方案：[⊖]

1）对请求操作提供及时的反馈：

- 及时确认用户输入。
- 对于花费大量时间的活动提供数据处理的进展指标。
- 提供最重要的信息形成初步的反应，当更多的信息变得可用时再将之提供。
- 需要应对复杂的请求时，提醒用户关注，并给予一个预期的延迟时间。

2）优化数据处理操作：

- 推迟低优先级的数据处理操作，直至计算资源变为可用。
- 尽可能预测数据处理过程中所需要的操作，并提前执行这些操作。

3）优化任务队列：

- 基于优先级重新排序任务队列。
- 刷新可能不再需要或已被取代的任务。

4）多任务性能监督：

- 监测多任务进展，调整资源分配，进而优化任务的执行和终止。
- 平衡任务持续时间和所需资源。
- 预测任务持续时间，并确定任务的不连续性、并发性和同步策略。
- 通过预测资源的冲突和死锁的情况，建立资源监测和监督调解机制。

2. 设计可靠性

软件的可靠性只能通过对每个可能的失效模式进行扩展设计评估来获得。这要求对每一个数据处理动作都需要检查其在何种条件、刺激或用户输入的情况下有可能带来异常的数据处理操作，导致错误的结果。这涉及计算环境中可能发生的事件；阻止外部系统或应用程序对关键数据的处理操作；或者是由某个接口系统或应用程序提供的错误的数据。

功能分析与分配（见第 11 章）期间所进行的失效模式和影响分析应该为决定如何对软件

⊖ 参见 http://blogs.msdn.com/b/zainala/archive/2008/08/21/tips-for-improving-software-responsiveness.aspx。

进行设计以对于潜在故障具备弹性兼容能力提供基础。在保障软件对故障的弹性能力方面，主要有两种途径：故障预防和容错。

故障预防建立了一种机制，旨在防止发生错误的输入或条件。故障预防直接影响结构设计方案，以确保数据处理事务中引入的所有数据都处于数据定义的可接受的预期边界内。这包括确保所有数据输入和用户界面的交互都处在系统监督之下，以防止引入不当的数据值。

容错是对错误条件的检测和故障恢复的操作选择。故障恢复有两种方法，一种可以返回软件之前保存的状态（向后恢复）；另一种是将多个功能与流程的结果进行比较，选择合适的结果继续执行（向前恢复）。如果涉及三个或更多的冗余进程，中间结果的比较（投票）可以确定哪些结果最有可能是不正确的，可以选择占主导地位的结果来进行数据的处理（屏蔽不正确的结果）。

在软件设计综合阶段，一项重要内容是要解决所需的故障检测和恢复的方案。故障预防会影响各个结构单元的规约。容错性将影响结构设计解决方案，在更大程度上满足冗余、并行的数据处理操作。容错是硬件产品的一种常见的工程设计概念，与冗余的组件并行工作，以在一个组件故障时确保运行性能不受影响。

3. 资源利用

软件设计方案必须对有限的计算资源提供有效的分配和利用方式。在软件设计综合过程中，一项重要的内容是要建立资源利用预算，据此可以建立结构的规约。这个预算应该提供对预期的资源消耗的描述，其中包括每个线程的行为概要。在必要的时候，需要为单个结构组件和单元建立进一步的资源预算。工程组套可能需要建立资源监控桩来捕获实际的资源消耗概要，为资源利用的优化提供基础。图 12-4 表示软件架构设计和实现流程，以确保资源利用的预算能够成功实现。

图 12-4　软件资源利用的实现

软件工程涉及很多与资源利用相关的设计策略，必须考虑建立一个高效、有效的设计方案。在功能分析与分配过程中，软件的行为应包含功能线程、各个组件、单元之间的资源分配。对资源实施监督的活动应纳入行为模型，以评估多任务调度、任务优先级以及资源排队策略。在设计综合阶段，以上所论的结果应该被纳入相关结构组件和单元的规约之中。资源攸关的结构组件应认定为工程组套，指定性能评估所需的资源利用桩。

建立软件架构之后，在适当的时候，计算资源的利用策略必须纳入软件设计和编码的标准。这包括任务优先级、多任务调度、排队和垃圾收集计划。生成的软件技术数据包、设计和编码指南应该为计算资源控制和保护的处理提供一个综合的蓝图。

12.2.6　软件原型

原型用于生成一个工程的仿真模型，其目的在于评估性能、可用性和与图形用户界面、图形渲染、数据交换的吞吐量或数据表示形式相关的美学特性，包括资料的印刷或绘制等。软件原型是一个普遍接受的实践，能够有效收集涉众对产品配置的部分反馈。然而，往往这些软件原型还需要通过迭代或螺旋方法才能演化成最终的产品配置。这是对公认工程原则所主张的原型实践的一种误用。

软件原型已经成为一种公认的软件开发实践，因为它使用并不高昂的成本，建造了一个与最终产品具有同样的方式和计算语言的原型。然而，快速原型通常在创建时没有严格遵守设计和编码的实践。这导致了原型配置没有充分考虑其所需承受的操作环境要求。因此，如果软件的开发策略中预想了从原型出发，通过不断进化得到可交付产品的过程，那么它绕过了软件工程的实践，就会导致本质上脆弱、不稳定和不可维护的结构配置结果。

传统的工程准则将原型作为测试项目或概念证明的方法，用于按照“形成，适应和功能”的方式辅助产品的评估。代表性的原型塑造了对产品的概念，并在如下方面有益于产品的评估：

- 确定用于制造产品的材料。
- 通过功能测试和性能测试对设计进行验证。
- 对于可用的组件，验证其质量和可行性，以满足工程规约所预期的操作和环境条件。
- 确认开发（构造、组装和集成）过程。
- 优化产品设计特性。

有许多形式的模型和原型可以用于产品的工程活动。原型是产品设计的一个精确近似，用于产品的制造、构造或实现之中。原型不是成形的产品，因为测试和评估工作的破坏性，许多原型最终只用于展示、收藏或者被丢弃。

软件原型必须努力协调关键的设计挑战，而这些挑战难以经由其他类型的模型、仿真、
222 或工程技术来解决。软件原型已经颠覆了正式的软件工程实践，有利于增量产品开发策略。当前软件原型设计策略包括 4 个主要形式的原型：[⊖]

1）快速原型——在非常早期的阶段，通过相对较短的调查之后，创建软件产品各个部分的工作模型。构建原型所使用的方法通常是相当非正式的，最重要的因素是完成原型的速度。以模型为起点，用户可以重新审视他们的期望，明确他们的需求。当这个目的已经实现时，原型模型就会丢弃，然后基于细化的要求正式开发系统。

2）演化原型——以结构化的方式创建并不断完善的非常健壮的原型。演化原型形成了软件产品的基础。它允许软件团队对原型进行修改和扩展，而这一点在需求和设计活动中是无法实现的。演化原型可能使用预定的操作环境来配置和演化。软件产品是永远达不到“完成”和“成熟”阶段的，因为操作环境在不断地变化。

3）增量原型——最终的软件产品通过将单个的原型逐步集成到一个整体的产品配置之

⊖ 参见 http://en.wikipedia.org/wiki/Software_prototyping#Types_of_prototyping。

中而获得。

4）极限原型——极限原型用于开发 web 应用程序。它建立了增量软件构建的三个阶段。第一阶段是一个静态原型，内容涵盖了从 HTML 页面到页面布局设计。在第二阶段中，HTML 页面是动态的，以支持网站导航。在第三阶段，事务处理功能已经实现。

围绕这些软件原型策略的主要误解在于，认为这种方法与工程相类似。软件原型在传统软件开发方法基础上，服务于一个明确的工程目的。当前软件产业事实上已经接受了一个业余的、演化的原型软件产品设计和开发方法。这种方法拖延了业务产品的开发，随着时间的推移，会导致成本超支和广泛的进度拖延，并带来许多不合格的、没有纪律的软件专家，最终导致项目失败或取消。使用原型，或者说滥用原型，有其缺点：

- 错误的假设。关注有限原型可能会分散软件分析师处理完整问题空间的注意力。这可能导致忽视可选设计方案，不完整规范的制备，以及软件问题空间的复杂性所导致的分析师全面理解的缺乏。
- 原型无法规模化。因为一个原型功能有限，对它的设计进行扩展并不一定能满足最初的客户需求。在许多情况下，脆弱的原型结构框架不能以增强的方式提供一个稳定的、长期的设计方案。
- 用户错将原型当作成品的近似。客户认为软件原型是一个接近完成的最终产品，而仅仅需要增强和优化。这让客户对于开发团队准备的交付最终产品的形式产生了误解。客户可能要求将原型产品引入运营之中，而产品尚未准备好这样的尝试。
- 开发者受原型所限。开发人员忠于他们花了大量的努力所开发的原型。这可能导致他们试图通过改造原型的方式来获得最终系统，尽管原型不是建立在一个持久的设计架构之上。（这表明一次性成型的原型，而非演化的原型，是首选的方法。）
- 过度为原型的开发花费时间。原型的一个关键特性是，它应该是很快完成。如果开发人员忽略这个事实，他们很可能试图开发一个太复杂、太昂贵的原型。
- 实现一个原型的代价。构造软件原型的成本可能超过从它的存在所获得的好处。最初的原型的目的是通过对原型的评估来解决工程设计的挑战。然而，随着原型消耗越来越大的软件开发预算，这可能成为一项难以放弃的重大投资。

有时，软件原型的开发工作应该委托他人。每个结构方案的评估可能会揭示某些技术挑战或风险，而这些可以通过对基于软件的解决方案开发原型来核实。考虑软件原型的开发，以下因素是必须考虑的：

1）每个授权软件原型是朝着解决重要问题所迈出的一步。开发一个原型所耗费的资源必须有所回报，那就是澄清客户需求或者提供设计挑战、风险的解决方案。

2）软件原型的交付方式必须与“工程”交付软件产品的方式相类似。原型的开发必须明确，对最终产品的某些特性的理解和特色的设想不是原型要解决或要协助解决的问题。原型的范围必须限定于工程正在寻求解决的问题。然而，基于软件工程实践，我们必须对试验性的原型给予足够的容忍。

3）原型开发工作不需遵守软件设计、编码和其他质量相关实践，因为这不会增加原型的价值。这使得原型团队能够节约某些只是阻碍原型开发却没有提供有意义优点的实践或过程。

4）原型必须受到正确的规约和测试，以确保它设计和实现的方式能够证实原型评估的结果。原型的测试应该关注于确认原型所代表的设计特点是否得以正确体现。原型测试的目的是确保原型能够提供必要的数据来解决原型的目标问题。

软件设计综合实践

本章阐述了建立软件结构配置和物理架构所需执行的设计综合任务。结构配置识别组成软件产品的结构组件和单元。物理架构表示结构配置及其相关的组套、工程图纸、模型和文档。这里没有使用术语架构设计，但是必须建立架构设计指导原则以指导结构配置的制定。通过迭代应用软件设计综合实践并结合其他软件工程原理和实践可得到物理架构。因此，软件工程是一门学科，可以建立一个完整、一致和实用的软件产品架构。

正如本章所讨论的，软件设计综合需处理建立软件产品结构配置和物理架构所需的任务。软件工程中的设计综合概念是由系统工程改造而来，其中考虑了软件产品的独特特征。系统工程中的设计综合应该应用于计算环境和软件开发后的过程的设计。不管怎样，这两种设计综合非常一致，都可以应用于非软件产品或过程设计工作。图 13-1 描述了应用于软件产品配置及其结构单元、组件和工程组套的软件设计综合任务。图 13-1 只是根据软件设计综合任务所在的主要设计活动对其进行了逻辑分组，任务本身并无先后次序之分。主要的软件设计综合活动包括：

图 13-1　软件设计综合任务

1）设计概念——通过持续的功能分析建立初始的结构设计概念。通过头脑风暴和积极合作，提出各种结构设计概念，其中包括从典型的、到进步的、激进的、创新的、开拓的和革命的结构设计模式。这个过程包括识别形成上层结构配置的抽象结构组件和用户接口机制及其编排和相互作用。

2）设计解决方案——对着重阐述结构配置的物理架构的持续评估和改进。设计解决方案有助于在通过推断（绘出结论或推理）导出设计方案时对软件设计特性、功能和性能特点的权衡。这包括对结构单元和组件的识别，它们代表构成更低层次结构配置的结构组块或材料。

3）设计评估——对演化的结构设计解决方案进行评估以确定其适用性和完整性。这些设计评估旨在确定任何与待实现的设计方案的能力及其在预期生命周期中进行扩展和加强的可扩展性等相关联的挑战。性能工程评估应为设计方案的性能是否符合性能基准提供适合的评估。

4）设计关系——识别可以弥合设计鸿沟并形成结构配置的集成层的集成结构组件。

5）设计表现形式——对设计解决方案的寻求做出总结并核准结构配置。识别结构组合，其中包括结构组件以及附加的软件测试桩以支持集成测试。完成与每个设计元素相关的工程图纸、图表和相关文档，为由设计过渡到软件实现做准备。

6）配置控制——将软件产品配置的技术文档置于配置控制之下并准备好技术数据包。技术数据包中包含规约、图纸、图表及相关文档，它们将被提交给软件实现团队，传达每个软件配置元素的设计特点。

228
|
229

这些活动旨在将设计综合任务组织起来，应对设计解决方案从初始概念到问题解决和验证的演化。设计概念化可在功能分析的早期阶段进行。这使得结构设计解决方案可以以自然、本能的方式获取抽象结构组件。这个活动意味着软件设计实践是一个富有想象力的、艺术的活动。设计决议支持权衡分析。使用权衡分析可以完善概念设计或从完整的功能架构中识别出结构单元。必须通过识别集成结构组件来整合所获得的设计结果中的概念设计和结构单元。设计相关性建立了结构单元和由集成结构组件来实施的抽象组件之间的结构关联。

本章其余部分将讨论每一个软件设计综合任务，并为如何制定软件结构设计提供一些指导意见。这些任务为软件设计配置建立了一个三层范式。第一层（顶层）是通过设计概念化活动获得的概念结构设计组件。第三层（底层）标识出了构成结构配置的基本结构设计元素（单元或组件）。第二层（中间层）弥合了基本结构元素与概念设计结构之间的鸿沟。图 13-2 给出了这三层范式的具体定义，为本章软件设计综合的讨论提供参考。

13.1　设计概念化

设计概念化包括致力于建立核心结构配置的初始设计任务。可以在功能架构的初始层成形之后执行这些任务。进行概念化设计要识别抽象结构元素，进行广泛的设计策划以解决关键设计挑战。这时无需对概念化设计元素进行过细的约定，因为随着功能架构的不断演化并提供额外的技术见解，概念将被持续修改和精化。最好推迟接受建立在演绎或推论基础上的设计解释。在有技术证据支持某个优先设计选择方案前，维持概念设计解决方案的抽象性是明智的。

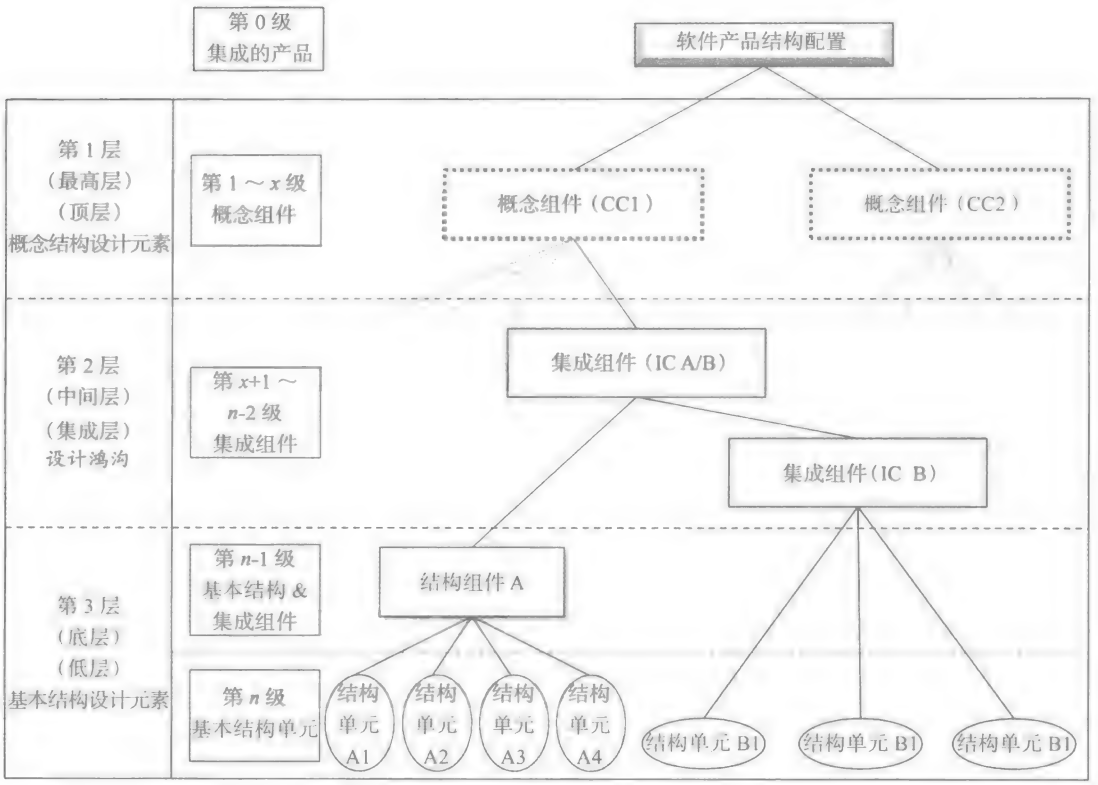


图 13-2　软件结构设计三层范式

230
～
231

13.1.1　建立软件架构设计指导原则

必须建立软件架构设计指导原则，指出如何设计或约束结构配置以促进有效、高效和可持续的解决方案的建立。软件架构指导原则给出了明确的设计范式，来指导结构配置的组织和安排，以及影响关于物理架构的设计决策。设计范式由一组互补的规则或原则构成，共同定义了一个决定性的方法来设计结构解决方案。

设计指导原则必须指出如何制定结构配置以建立一个确保设计完整性的架构框架。“架构稳定性评估”任务为设计完整性提供了额外的资料。制定架构指导原则时需要考虑以下准则：

1）架构持久性——结构指导原则旨在建立一个架构框架，该框架能够在其整个生命周期容纳产品的修改和增强。持久性意味着基础设计结构不发生变化。

- 配置健壮性——设计规则或策略旨在确保结构元素和设计机制可以进行修改、扩展或增强而不会破坏架构的完整性。
- 架构韧性——设计规则或策略旨在确保无论架构做了什么修改和增强，结构元素都能正常执行其数据处理操作。

2）架构简单性——旨在确保那些与理解数据处理规则复杂性相关的结构元素的安排和相互关系的结构指导原则。

- 元素复杂性——旨在确保计算准确性在分布结构元素间均衡分布的设计规则或策略。
- 集成密度——旨在确保集成任务中结构元素的均衡分布的设计规则或策略。
- 交互复杂性——旨在限制结构元素之间的交互（交互数据交换或控制传递的数量）的

设计规则或策略。

3）操作持久性——旨在确保在压力和破坏性的条件下数据处理操作的持续性的结构指导方针。

- 操作负荷弹性——旨在确保在计划内或极端负载条件下架构的性能一致的设计规则或策略。
- 操作中断适应性——确保架构能够承受与计算环境或外部系统元素有关的外部故障的设计规则或策略。
- 计算机技术适应性——确保架构对计算环境设备、系统或应用程序的变更具有弹性的设计规则或策略。

232

13.1.2　识别抽象结构组件

进行概念设计始于对抽象结构组件的识别。抽象组件的顶层代表软件功能的主要容器。这些抽象组件提供了设计概念表达、分析和讨论的基础。关于设计概念适宜性的审议会使概念变得更加合适。必须对由功能分析所得的最初的设计概念进行重新评估，以确定它是否体现了软件产品操作的性质。

抽象结构组件应按照它们操作行为主要线程的角色来标记。一个抽象组件应该是结构设计概念中一个单一的、集中的角色。然而并没有限制一个抽象组件可以承担多少个角色的责任。在考虑如何标记一个抽象组件时，应主要聚焦于抽象结构组件如何搭配能方便其交互从而减少结构的复杂性。

应该用软件框图来表示概念设计层，描述抽象结构组件的安排及其接口（内部和外部）。因为软件配置不包含任何物理维度，抽象组件的结构布局应该基于设计概念中各组件的相对关系。应该定义抽象接口来初步表示抽象组件与外部系统之间的数据交换关系。软件框图应从不同的考虑层次展现概念化设计，以应对最决定性的设计挑战。

在形成概念设计的初期阶段没有必要考虑每一个想得到的抽象组件。概念设计只是一个组织结构配置从而反映从功能解决方案中所得到新认识的方法。它有利于制定几个设计概念分别强调不同的架构设计方案，并从中选取最适合的概念设计配置。需要对这些备选的概念配置进行评估，根据功能架构、涉众需求、项目成本和进度限制，找出其中最平衡的解决方案。

13.1.3　识别抽象用户接口机制

用户接口机制，以屏幕、窗口、显示小装置的形式，往往形成软件概念化设计配置的焦点。软件设计概念可能包括能被视为结构组件的抽象用户接口机制的识别。用户接口机制通常提供对软件功能的访问和激活。然而，用户接口机制（如窗体、屏幕、对话框、下拉菜单或按钮）是其行为有助于用户与软件产品交互的结构设计部件。

233

在设计解决方案的概念化阶段，用户接口机制本质上应该是抽象的，为下一步的设计描述提供占位符。概念设计方案不应提出用户界面图、模型或原型，因为在此阶段确定设计偏好可能还为时过早。必须形成用户接口机制，以帮助用户与软件产品的交互，因而需要考虑应用人的因素和人体工程学[⊖]分析（身体、认知、组织和环境）以便人和软件能够适当的交互。

⊖ 访问 *http://en.wikipedia.org/wiki/Human_factors_and_ergonomics* 可获得对认知经济学形式的人 - 系统 / 人 - 机交互的更多了解。此处使用术语人 – 软件交互阐述影响软件产品设计的因素。

随着结构设计解决方案的改进，可能需要强调用户界面的设计，以便涉众对机制、编排和外观的合适性达成共识。为了使得用户接受所提出的用户界面设计方案，最后一招就是对用户界面建立原型。开发用户界面原型非常有用，它可以为用户提供一个有形的、可见的模型或者可视化的图形显示机制。每一个原型都是项目的一份额外投资，为描绘用户界面的可用性设计方案提供了另外的方法。原型开发与软件开发的工作类似，都要指定范围、设计架构和实现（设计、编码、测试和集成）。

在选择进行费时费力的用户界面原型开发之前，应该先考虑采用以下类型的设计表达方法：

1）说明：通过图表或者计算机生成的图形来说明用户接口（UI）机制。

2）内容映射：标识出 UI 机制之间的关系和跳转方式。此方法在表达数据输入输出表单布局方面非常有用。

3）故事板：用来表达 UI 屏幕进度从而展示用户的体验。故事板可以草拟与单独操作流程相关的 UI 设计概念。

4）建模：采用自动化设计工具生成 UI 机制或屏幕，能够快速生成和修改 UI 图形。利用 UI 模型可以以低廉的成本获得涉众反馈，而这些反馈会有益于 UI 设计机制规约的撰写。
234 最终的 UI 设计文档和图表也可以通过工具集自动生成。

13.2　设计解决方案

设计解决方案包括了对基本结构设计元素和配置选项的识别，以调查每个结构设计解决方案的特点。要使用整个功能架构，必须识别和指定结构单元和组件。结构单元提供了执行指定的功能行为以及形成结构设计解决方案所需的“材料”或构件。

13.2.1　识别基本结构元素

必须识别基本结构设计元素以建立结构设计方案的底层。基本结构设计元素包括结构单元和组件，它们在图 13-2 所示软件结构设计范式的底部描述。通过将相似、兼容或者互补的功能单元分组和合并成一个单一的结构设计元素，可以建立和合成结构单元。结构设计方案的最底层的结构组件代表复杂一些的功能单元，其行为解决方案是可理解的，并且不需要进一步分解。为了提供一个高效、有效的设计方案，这些结构组件也可能需要分解成两个或两个以上的结构单元。

完成了“确定基本结构元素”的任务后就会得到底层的结构设计元素。理想的底层结构不超过两层元素：低层的结构单元和上层的组织结构组件。这些基本结构元素代表了软件模块——将会说明并传达给软件实现团队以进行编程（详细设计、编码、测试和集成）的过程或对象。

通过对功能单元进行排序、关联并将之组合为一个内聚和综合的设计元素可确认结构单元。这通过如下步骤完成：

- 将多个功能单元规约统合成一个结构单元规约。
- 在统合的规约中消除矛盾和不一致的特性。
- 提炼结构单元需具有的性能属性。
- 将必要的监督或管理行为纳入规约。

由此产生的结构单元规约应包含阐述结构单元内聚行为的需求。

结构组件是通过分析功能组件规约和确定满足功能组件行为所需的先决结构机制而获取 235
的。这将导致至少识别出两个或两个以上的结构单元，把它们集成在一起，仍遵从功能组件规约。

13.2.2　识别集成组件

设计综合实践进行到此时，通过对结构设计元素的概念和基础的组织，已解决了设计方案的两个层次。必须弥合这两层之间的设计鸿沟，将概念设计层和结构设计层联合起来。通过如下两种方法识别集成结构组件来弥合此设计鸿沟：

- 结构设计层逐步延伸到概念设计组件。
- 逐步分解概念设计组件，并封装成结构设计层的元素。

组件集成使概念设计层和结构设计层得以合并成一个全面的、统一的结构设计解决方案。组件集成应该作为概念机制，其目的是提供一个组装好的组件。集成层提供了一层或多层的集成组件，将底层的结构组件与上层概念组件连接在一起。以下对如何识别集成组件提供了一些原则性的建议：

- 集成组件应该为结构设计层到概念设计层提供一个直观的过渡。
- 每个集成组件应该代表一个明显的组装和集成行为，并在结构配置中形成一个可识别的元素。
- 集成组件应为软件的集成和测试策略提供一个明确的进度说明。每次集成和测试的结果都应是具有已验证接口的自包含且已证明的结构组件。
- 应避免微小的、增量集成行为，除非是为了降低风险而特意设计的。

每一个集成组件都构成一个更高层次的结构组件。区分结构和集成组件的方法是根据它们在结构配置中的位置。结构组件位于最低层的上部，由功能架构明确导出，是大型、复杂的功能单元。

13.2.3　评估软件重用机会

随着结构配置的成熟，调查现有的软件包的重用机会可能是有利的。现有的软件包可能 236
是其他公司开发的，或企业内部软件存储库中原有的。必须通过工程成本效益分析来确定将现有软件包纳入软件产品配置是否有益。这有时被称为“自制或购进”分析，用来支持设计决策。自制或购进分析会对利用软件非开发产品（non developmental item, NDI）的可行性进行评估，并与开发新软件包（结构组件或单元）的费用进行比较。自制或购进分析是一种权衡分析，将在第 14 章讨论。

对一个软件进行自制或购进分析可以评估将现有的软件包并入设计结构的优势。在此分析中应明确以下因素：

1）合适的 NDI 解决方案的可用性。软件包可能具有很多与结构元素定义一致的表面特征。然而，以下几个产业实践可以确定是否重用现有软件包：

- 什么是 NDI 软件包的成熟度？根据软件包的操作经验，它可靠吗？根据 NDI 软件包的问题报告和修复情况，它稳定吗？
- 如果将软件包纳入软件产品配置，其提供者或供应商是否会继续为它提供支持？
- 如果其供应商倒闭，能买到它的源代码和文档以供维护吗？
- 将 NDI 软件包作为软件产品配置的元素，是否存在许可证或专利分配问题？

2）NDI 包的技术特点。尽管一个 NDI 软件包第一眼看上去很符合规约，但还必须考察它的性能相关的细节。

- 在特定的计算环境下，NDI 包是否能正确执行？
- 在与目标计算环境相似的计算环境下，NDI 包的性能基准是什么？
- NDI 包利用计算资源的效率如何？
- 数据计算的精度如何？
- 实现 NDI 软件的语言是什么？是否可以把另一种语言整合到整个软件产品配置中去？NDI 包可以组装到软件产品配置中并与其保持一致吗？或者 NDI 包是否需要在编译的时候才连接？
- NDI 包的外部接口和相关的软件接口是否有充足的文档，以便将包集成到软件产品结构配置？

237

对每一个候选 NDI 包，在纳入软件设计配置时，软件工程团队必须收集其可用性和适用性信息。这些信息可用来进行软件重用权衡分析。针对与对应软件结构单元或组件的进一步开发相关的实现和测试成本，对候选 NDI 的成本、效益、性能和风险进行评估。使用商用数据库管理系统（DBMS）或类似的 COTS 包不需要进行自制或购进分析。这些软件包广泛应用于软件应用程序开发，已经证明了它们在数据保存任务上的适应性。

13.3　设计相关性

设计相关性应用软件工程实践调整结构配置，以获得规约的设计方案。内聚的结构设计展现了必须根据功能架构和软件需求规约进行测量的协作技术和性能特征。必须调整设计解决方案中缺乏这些先决条件的方面，使得结构设计符合要求。设计改进的活动应在工程权衡分析的指引下进行。

13.3.1　建立性能基准

应该为结构设计解决方案的底层建立重要设计机制的性能特征（基准）。重要设计机制中结构元素的概念化数据结构、数据转换算法、数据完整性保证和数据转移过程在计算上可能是高要求或挑战性的。性能基准在结构设计的边界内建立起对设计机制的数据处理效率和有效性的估计。性能基准为结构元素建立了设计目标，在此基础上可以对实现进行评估。性能基准由对数据处理功能在结构单元和在结构配置的底层识别的组件中的分布做出说明的结构规约导出。

必须为底层结构元素建立性能基准，因为它们代表了根据结构规约实现（设计、编码和测试）的基本构件。性能基准必须为更低层的结构单元和组件建立性能需求。

集成软件产品的性能是对结构元素多层次顺序集成的结果。软件性能工程必须在集成组件或产品级对任何集成负担造成的后果进行说明。必须考虑贯穿集成层次的渐进同化从而建立准确的结构元素性能规约。

238

在集成前的初始评估中，计算资源密集型过程可以充分执行。然而，由于集成后必须与其他数据处理线程共享计算资源，结构元素性能会出现集成减损。应了解相关的、相互依存的和独立的设计元素之间计算资源分摊的含义。这表明软件执行情况必须在计算系统执行框架下分析。这需要软件性能基准对资源共享、多用户工作负载和软件产品操作过程中可能遇到的其他条件做出说明。

软件性能基准应该为结构元素在整个结构配置中建立计算时间和资源利用的需求。这些基准必须推断结构设计机制性能，对受到共享计算系统的资源需求和分配策略影响的集成产品的执行情况进行说明。这些性能基准为功能架构性能特征提供了工程近似法。

这种做法的目的是确保在结构配置中考虑到软件产品性能的设计。把对软件性能的关注推迟到测试阶段是不可接受的。到那时，架构决策已建立了设计配置，此配置可能会阻碍性能的满意度。软件性能必须是一个贯穿软件产品工程过程的整体的设计思路。软件性能基准的建立指明了对于结构设计元素如何提供可架构、可协作的数据处理机制以满足性能目标的理解。性能是通过对下至底层设计元素、上到整个结构配置的所有性能相关的度量进行聚合得到的。

13.3.2　识别结构设计缺点

应该对结构配置进行评估以识别建立一个完整的、一致的设计方案所面临的设计挑战和阻碍。识别出的结构元素包括可能无法充分支持有效或高效的数据处理事务的抽象设计机制。必须强化这些抽象设计机制，或者用那些在任何业务场景和条件下都能正确执行的设计机制来替换它们。应评估识别出的设计问题，并确定其优先级：

1）必需解决的技术问题。

2）关联到整个设计方案的相关范围。

3）如果缺陷没有解决，其后果或影响。

设计问题的优先级将软件工程团队的注意力集中于对设计方案的影响最大的设计问题 239
上。许多设计挑战可能会对项目的成功实施构成风险。风险涉及结构设计的任何方面，可能会对以下项目的成功因素造成潜在影响：

- 软件开发项目有充足的预算、资源和时间，能够为项目成功执行提供很大信心。
- 产品的适用性和可靠性能够满足涉众的需求和期望。对于消费品，这表示产品的可用性能够满足市场并产生预期的投资回报。
- 软件开发团队拥有可用的技术技能和专业知识，能够成功实现和测试结构设计机制。
- 结构设计为软件产品维护（开发后期）提供了有效的基础。这包括结构配置及其设计机制，能够促进：1）识别、隔离和解决设计缺陷（bug 修复）；2）融入预先计划的产品改进（preplanned product improvements, P^3I）。

必须评估那些优先的设计缺陷以确定需要对哪些设计问题进行进一步探讨，以改善设计解决方案。简单地按照软件设计的影响来分类，可以划分为 4 个主要领域。下面先介绍这 4 个主要领域，以便引入后续的设计综合任务：

1）*产品设计的偏好*。根据性能特征对可选设计方案进行评估。这个任务主要是评估每个提出的设计策略的有效性、效率和简洁性。通过这个任务可以对候选方案按技术偏好排名。

2）*产品实现的影响*。根据软件实现和测试组织将抽象设计机制转换为结构设计解决方案的能力，对可选设计方案进行评估。

3）*产品维护的影响*。根据软件维护组织对包括抽象设计机制的结构设计解决方案进行修复、扩展（增加范围）、加强（包含额外的功能或数据处理变化）的能力，对可选设计方案进行评估。

4）*产品设计的完整性*。根据包括抽象设计机制的结构设计解决方案的结构完整性和完整性相关属性（对设计原则和标准的坚持）、对架构级结构进行评估。

13.3.3　评估架构候选方案

240 架构候选方案以独特的抽象设计机制标识出要执行的数据处理事务。在没有足够的设计分析的情况下，可能很难完美解决复杂的设计状况。头脑风暴可以用来识别潜在的可选设计，并导向创新的解决方案。然而，必须在对每个设计范式或机制的优缺点有足够理解的情况下做设计决策。首选的架构方法应该能够平衡设计有效性、效率、可伸缩性和简单性。建立架构质量因素或者偏好的相对重要性，是软件产品设计被错误地当作一种艺术化实践的地方。术语架构在建筑工程学科中包含了艺术与科学的统一。如果产品的性能不能满足客户的操作要求或业务环境的要求，无论多么有吸引力的用户界面都是无足轻重的。

应对所有的可选方案进行权衡研究以选择一个最好的架构方案。每个权衡研究都源于有两个或两个以上的方法可以解决某技术障碍。这些相互竞争的可选设计方案必须是已定义并且充分支持全面的权衡分析的（见第 14 章）。

13.3.4　评估软件实现挑战

软件工程团队包括确保结构配置和集成为软件实现提供上下文的软件实现团队代表。软件实现团队代表必须认可结构设计解决方案在已建立的实现计划、资源和进度约束内能够完成。软件实现团队必须评估架构解决方案所蕴含的工作量范围。考虑软件实现工作的工作量时必须考虑以下几个因素：

1）软件实现人员的技能和经验，包括将结构元素转换成一个完整的软件产品配置的设计、编码、测试和集成的技能和经验。

2）编程语言构造（语句、语义、调用关系、可扩展性等）的数据操作灵活性，使软件单元的设计和编码有效且高效。

3）架构规约具有适当性和可适应性，便于实现。

4）分配给软件实现任务的项目资源（人员、设备、设施、资金、进度等）的可用性。

在关键设计评审之前必须解决软件实现的挑战。CDR 代表了一个项目的里程碑，标志着架构解决方案已经足够开始软件实现阶段。软件实现团队应该在 CDR 执行之前就能够支
241 持架构解决方案。软件实现团队在架构解决方案中任何挥之不去的问题都不应推迟项目架构解决方案的审查。

通常，除了技术挑战的原因之外，编程语言的选择都是由架构方案决定的。通常情况下，编程语言的选择都是由经验丰富的熟练员工决定的。然而，编程语言的选择可能会给架构解决方案的实现带来重大挑战。进行编程语言选择时必须考虑下列因素：

1）编程语言技术能力。编程语言是否支持架构解决方案的数据处理特征？

2）程序员生产力。软件实现团队是否精通编程语言，能够应付架构解决方案的设计、编码、测试和集成？

3）编程语言工具的可用性以及相关培训和咨询服务。该编程语言是否有足够的教育和技术服务，以促进员工知识获取和自动化工具支持。编程语言工具应包括编译器、汇编器、编程设计、代码生成、调试和文档化的应用程序。

13.3.5　评估软件维护挑战

软件工程团队中要有软件维护团队的代表，以确保架构配置和设计机制提供了对客户和产品支持的背景。软件维护代表必须认可结构设计解决方案，声明在已建立的维护计划、资

源和进度约束范围内，它是可维护的。

软件产品维护的基线工作计划可能包括提供客户支持、帮助平台、设计缺陷决议和 P^3I。一旦最初的软件开发工作完成，软件维护团队中就可以增加权威的软件工程和实施人员，监督软件产品的提升。

13.3.6　评估架构完整性

必须评估物理架构的完整性，以确定其适应未来变化且符合已建立的架构指导原则的能力。以下是根据 13.1.1 节中的指南所提出的一些评估定义，以建立架构设计指导原则：

1）架构持久性——评估架构框架在其生命周期中容忍产品修改和增强的能力。 242

- 配置健壮性——在不违反架构指南的情况下，修改、扩展或增强结构元素和设计机制的能力。
- 元素持久性——无论架构如何修改或增强，其结构元素执行数据处理操作的能力。

2）架构简单性——评估与理解数据处理约定的能力相关的结构元素的安排和它们之间的相互关系。

- 元素复杂性——评估计算在结构元素中的分布。
- 集成密度——评估集成任务中结构元素的均衡分布。
- 交互复杂性——评估结构元素之间需要交换或传递的交互数据的数量。

3）操作持久性——架构在压力和破坏性条件下维持数据处理操作的能力。

- 操作负荷的弹性——评估架构在计划工作负载和极端负载下性能的一致性。
- 操作中断的适应性——评估架构承受与计算环境或外部系统元素相关的外部故障的能力。
- 计算机技术同化——架构适应计算环境的设备、系统或应用程序变化的能力。

这些架构评估关注结构配置承受由操作需求的变化、技术的改进或设计缺陷的解决等所造成的改变的能力的三个主要方面。架构持久性和简单性标志着结构配置的优势所在，并使之能容忍开发后的变动（产品操作和维护）。设计的简单性与复杂性正好相反，是通过采用少量的简单元素达到最大效果。这是通过追求极简架构得到的。设计持久性标志着产品设计的结构基础的本质。持久性的反面是暂时性，暂时性是指架构只能维持很短的时间，不能在较长一段时间内保持性能不变。架构的持久性和简单性设计必须建立在一个稳定的架构配置之上。

操作持久性标志着产品架构适应预期运行情况和适应计算技术变化的能力。这些操作持久性度量标志着软件架构自我调整应对操作环境或状态变化的能力。目的是无论软件产品遭受什么样的变更、不稳定或什么样的操作条件，软件产品都能将性能维持在可接受的水平。 243

13.4　设计表现

设计表现是采用架构图表、图纸和文档对结构配置的详细展现。包括结构设计中每个元素、内部和外部接口和相关的数据结构的规约。需要软件桩来支持软件集成测试的结构配置项应被识别为工程组套和测试桩。

13.4.1　建立结构设计配置

结构设计配置应置于技术配置控制之下，以防无意中引入更改。结构配置的每个元素都

应通过认可的软件配置控制程序来唯一确定。此后，只有通过软件变更控制委员会（CCB）批准的变更请求或提议才能集成到结构配置中去。

13.4.2　说明结构配置元素

必须对每个结构配置的元素进行规约说明以支持软件实现。这些规约代表了结构单元和组件进行设计、代码生成、测试和集成的技术要求。每个结构元素规约及其图表在成为软件技术数据包之前，应置于技术配置控制之下。

13.4.3　识别工程组套

应当识别工程组套及其附加的测试桩。工程组套的识别为软件实现团队计划和执行软件集成和测试提供了完整的工作范围。工程组套包括集成的结构组件及其用于验证集成的相关测试桩程序。

13.5　准备软件技术数据包

软件技术数据包必须在准备 CDR 的过程中完成。在开发的这个阶段，软件技术数据包被标记为工程进行中，以便它与支持软件构建和产品复制过程的发布状态区别开。TDP 中包
244 含软件实现阶段的软件物料清单。它确定了描述软件产品开发的架构材料。TDP 包括要开发的软件产品的全套图纸、图表、文档和模型。TDP 包括了软件 BOM 和整套软件产品文档，或者是每种材料授权版本的参考物。拥有的物料项若是电子的，则必须存放在工程数据管理中心（或软件产品数据管理（PDM）应用程序）中，并标识文件标识符和存储位置。

软件物料清单中必须确定包含以下材料：

1. 软件产品标识
    1.1. 命名法
    1.2. 产品标识
2. 软件需求基线
    2.1. 软件需求规约
    2.2. 软件接口规约
    2.3. 业务模型
        2.3.1. 业务场景 A
        2.3.2. 业务场景 B
    2.4. 业务环境描述
3. 软件功能架构
    3.1. 功能层次
    3.2. 行为模型
    3.3. 功能规约（可选）[⊖]
4. 软件物理架构
    4.1. 架构指导方针和原则

⊖ 注：软件 BOM 中已包含功能架构以提供软件产品架构的完整描述。功能单元规约被整合到结构配置元素规约中。因此，软件 BOM 中就没有必要再描述它们，尽管软件配置审核中还需要他们的支持。

4.2. 结构配置——顶层（抽象组件）
4.3. 结构配置——2-*N* 层（集成组件）
4.4. 结构配置——*N* 层（结构组件）
4.5. 结构配置——*M* 层（结构单元）
5. 外部接口设计
5.1. 数据库设计文档
5.1.1. 表描述
5.1.2 查询描述
5.2.（外部系统）接口描述文档
6. 重要的变更请求和提议
7. 计算环境描述
8. 备注

245
|
246

第 14 章

Software Engineering: Architecture-Driven Software Development

软件分析实践

软件分析实践阶段涉及旨在执行各种工程权衡研究的分析任务以协助进行基于架构的设计决策。软件分析权衡研究针对复杂状况的处理涉及与软件产品及其生命周期，以及软件开发项目等有关的很多因素。软件分析权衡研究只注重建立软件产品特性，这也使得它区别于其他类型的分析技术。

软件分析权衡研究也被称为权衡分析，是一种成本效益分析技术。工程权衡研究比成本效益分析更为复杂，因为它要在竞争产品的特性、技术、项目相关风险因素以及生命周期成本影响等更多领域中取得权衡。软件架构候选设计方案必须符合涉众的满意度水平，可计算软件生命周期成本（包括开发过程中以及开发后期相关处理工作），并能预计随时间推移可为企业获得的利益。由于在决策制定阶段考虑了全方位因素，因而软件工程权衡研究考虑得更为整体全面。

权衡分析必须提供一种量化评估方式来比较架构候选方案。架构决策应有助于为软件产品建立一个持久的结构化框架（组成、组织、安排和结构），使得软件产品能够承受极端操作情况，适应不断变化的计算机技术，并适应日后的提高和改进。软件架构候选方案的评价必须经由分析师收集各种对设计有影响的信息，包括：

- 软件产品特性和功能。
- 产品性能。
- 美学或用户界面的“外观和体验”。
- 操作难易程度。
- 测试和评估的难易程度。
- 对软件复制（在分发媒介上重现可执行程序）和发布的影响。
- 对用户培训和理解的影响。
- 对客户支持过程的影响。
- 对产品支持过程的影响。
- 对产品提升和可伸缩性的影响。
- 结构目标的协调，例如产品线、组件重用以及产品框架。

软件分析包括 16 个任务，共分为 6 大主题。这些主题体现了一个典型的权衡研究工作流程。然而在每个主题内部，只要适合当前的权衡研究环境，这些任务可以以任意顺序执行。权衡研究的主题和任务见图 14-1。6 个主题是：

1）定义权衡研究。

2）准备权衡研究环境。

3）实施评估。

4）评估项目影响。

5）评估权衡研究结果。

6）决策统一。

248

图 14-1　软件分析任务

249

14.1　定义权衡研究

每个架构的权衡研究都需要正式经过软件工程团队领导的授权。架构权衡研究中，一个架构功能点、差异化的特性或者性能目标都可能严重影响产品性能、质量或耐用性。设计方案不应由个体单独决定，因为某个决策的影响可能将贯穿该产品的整个生命周期。因此，决策必须经过软件工程团队的一致同意而产生，也可参考项目管理团队其他成员、客户以及涉众的意见。

14.1.1　建立权衡研究领域

权衡研究源于架构层面的挑战对软件产品定义的影响，包括如下领域：1）建立一个可实现的软件需求基线；2）制定功能架构以及典型行为模型；3）为物理架构建立健壮的结构配置。每个领域代表都代表一个不同的架构决策，这些决策可能潜在地影响架构解决方案的其他方面。

每个权衡研究领域必须能够对调研起到约束作用，并确保为制定健全公正的设计决策提供必要的数据支持。因此实施权衡研究需要考虑周全。权衡研究领域的建立由采集和分析的数据驱动，旨在帮助架构评估。所要调研的问题可能涉及客户需求、性能、行为、结构或质量等方面。然而，每个架构决策可能同时与产品定义和其生命周期过程的其他方面相关。不能随意地决定任何一个架构决策。相反地，在评估架构中存在的重要问题时，也应缓和而谨

慎地进行权衡分析。

14.1.2　确定候选方案

对于每个架构难题来说可能都存在许多种候选方案，这使得从中选出一个可行方案变得很困难。权衡研究只考虑那些对软件架构有较大改进和提升的候选方案。因而原先的候选方案列表被大大缩减，形成数量较少的可行方案集合。应该对候选方案进行评估，以推测将该方案融入软架架构中的预期效益和结果。在尝试评估候选方案可行性时应考虑如下问题：

1）与解决方案相关的重要的技术特点是什么？

250

2）该解决方案对软件制品的架构完整性（依照架构准则）有何贡献？

3）该解决方案对集成软件解决方案的性能特点有何影响？

4）对于实现和测试该解决方案的难易程度，如何定义其可接受水平？

5）该解决方案对软件的操作和维护过程有何影响？

上述问题以及其他问题可以作为权衡研究中判定候选方案的基本标准。该基本标准提供了候选方案可进入可行方案集合所要满足的必要条件。在进行权衡分析前，应该对可行方案进行优先级排序以建立一个候选方案的权威优先排名。优先候选方案列表需要在权衡分析的启动会议上就出示给相关涉众代表。涉众代表们也需要慎重考虑其他可能影响候选方案排名的因素。

14.1.3　建立成功标准

建立成功标准才能对候选方案进行评估。成功标准包含描述理想方案的最低限度可接受的客观因素。每个因素都有一个取值范围，候选方案按照对该因素的符合水平被赋予相应的取值。被选为最适合应用到软件架构中的方案必须符合大部分成功标准。

成功标准必须按照软件生命周期的各项因素进行分类，包括性能、技术实现难易度、可用性、软件架构健壮性以及项目成功风险等。对这些标准进行加权来建立一个平衡的评价体系。这些成功标准以及加权机制必须保证所有成功因素都能帮助筛选出更优的解决方案。

雷达图或蜘蛛图可以提供图形化手段来描述成功标准以及候选方案排名。雷达图可以在3个或多个变量的二维平面中展示多元变量的数据。每个变量或成功因素在从中心点向外扩散的轴上进行度量，类似于车轮辐条。最里面的环表示每个成功因素的最小取值，最外面的环表示目标值。当某个因素的度量值低于最里面的环，接近于中心，表示该因素未达到最低标准。如果度量值超过最外面的环，表示该因素已经超过了目标值。在雷达图中可以显示多个候选方案以便进行比较。图 14-2 显示了一个雷达图示例。

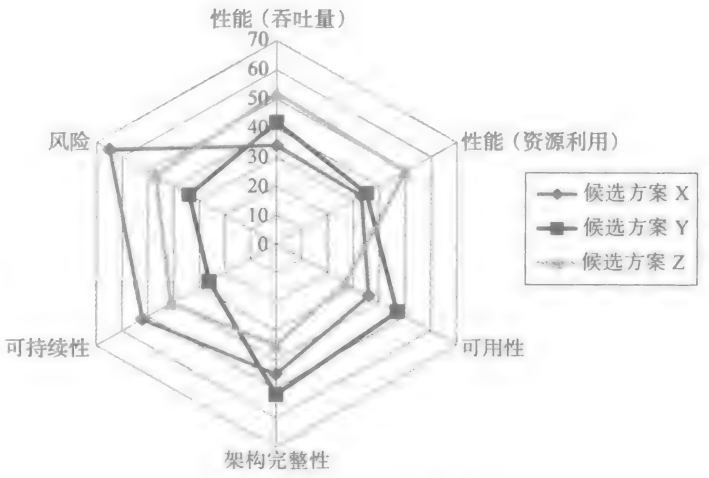


图 14-2　雷达图或蜘蛛图示例

14.2　建立权衡研究环境

必须建立良好的权衡研究环境，以帮助相关人员展开调研工作。权衡研究环境涉及相关机制（工具、设备和场景）、数据收集和分析工具，以及执行权衡研究的流程步骤。

执行权衡研究有多种方法。“实验设计”是指用一个正规而科学的方法在受控条件下收集数据。此处的实验是指用高效的手段来收集工程数据，用以得出有效结论。要建立一个“受控的”实验，需要保证在该实验环境下能得到一致的、可重复的实验结果。权衡研究方法必须确保对竞争架构候选方案的正确和公正评估。

14.2.1　汇集实验机制

权衡分析需要提供与调研相关的充分的信息，来帮助确定所需的相关工具、模型或仿真方法。这些实验机制需要进行收集和限定。权衡分析的工具以及模型可以为架构候选方案提供抽象化表示。计算机仿真可提供动态建模机制，来模拟这些模型在接收刺激或在特定条件下的操作反应或行为反应。实验机制需要保证每个架构候选方案都在相同条件下进行评估，以确保收集数据的一致性。

251
|
252

实验机制需要展示架构候选方案的一些工程细节，用以反映整个软件架构的成熟度。该机制需要体现出与其他候选方案相比具有竞争力的设计特点。实验技术必须适用于其他候选方案并生成可供比较分析的数据集。场景则需要指出实施实验的前置条件和事件序列。不同于设计测试用例，此时不用指出场景的预期结果，而是通过获得的实验结果来分析该架构候选方案与场景如何进行交互。

14.2.2　汇集数据收集和分析机制

权衡分析的目的是确定每个架构方案的相对价值。为了达到这个目的，获取和分析数据的能力对最终产生的结果的相关性和适用性具有至关重要的影响。数据收集工具、仪器及相关物品根据实验机制进行预处理。因此，制定实验机制的同时也需要考虑数据收集和分析机制。

目前有多种实验数据采集和分析方法，它们依赖于所采用的实验。表 14-1 列出了可用的数据采集机制以及对应的分析技术类型。当采用计算机建模或模拟时，需要度量每个模型的保真度来获取和保存那些最相关的数据元素。模型参数的精度必须统一来确保数据集的一致性。经过实验得到的数据应存储妥当，作为权衡研究结果的历史记录。表 14-1 所示的数据分析技术定义如下：

1）时间轴分析——对一系列数据点的分析，尤其是对一个连续时间段等间隔取样的数据点。按时间顺序进行分析，有助于从数据集中获取有价值的统计以及其他特性。

2）数据挖掘——一种从海量数据集或数据库中发掘数据间的模式的技术。

3）敏感度分析——探索在不同输入假设和输入场景下，模型的输出与输入有怎样的依赖关系。

253

4）结构分析——确定物理因素和环境因素对架构的物理结构及其组件的影响。建筑、桥梁、车辆、机械、家具等常需要结构分析。分析结果用来判断该结构是否适合使用，经常可减免物理压力测试的需要。

5）行为分析——重现模型的某些行为，包括功能流、控制流和数据流，资源利用率，执行时间，接口传输速率。

6）定量分析——通过复杂的数学和统计方面的建模、度量和研究分析来提高决策的整体质量。这些技术通常用于功能模型或行为模型、决策树以及模拟器。

7）定性分析——一种不使用数学分析和统计分析的较为主观的数据分析技术。该技术通过研究系统所表现出来的行为，来理解其背后的设计思想。

8）仿真——随时间对实际过程和系统的行为的模仿。仿真通过自然或人为系统的工程模型，在不同条件下观察它们的性能和行为。

表 14-1　数据采集机制和分析技术

数据采集机制	说明	数据分析技术							
		时间轴	数据挖掘	敏感度分析	结构分析	行为分析	定量分析	定性分析	仿真
调查	适用于需要对该领域提供专业意见的人	×	×						
面谈	适用于希望从涉众、用户或专家取得第一手资料的人，不适合为了收集数据而访问大量人群	×	×						
小组共识	在特定小组的组员当中达成共识并整理大家的意见和判断	×				×		×	
工程评价	用专业的工程知识来制定解决方案或行动方案	×	×	×		×		×	
科学实验	在受控条件下进行实验，来阐述公理、检验假设的正确性或判定未被执行过的方案的有效性			×	×	×	×	×	×
物理模型	对实际研究的产品或过程的概念描述、图形化描述或数学描述。当无法用科学实验得出结果时，常常会采用物理模型。模型也可以帮助人们描述思维过程或假设	×		×	×		×	×	
计算机模型	对产品或过程构建软件模型，有助于理解预期的或非预期的行为和结果	×		×	×	×	×	×	×

14.2.3　建立权衡研究过程

权衡研究属于受控实验，遵守统一流程。该流程需要建立一个可以满足所有实验条件的应用并对每个候选方案进行评估。权衡研究流程分为如下阶段：

1）建立环境——为候选方案准备实验环境。

2）初始化——加载场景数据集，制定数据采集机制来获取和记录实验相关数据。

3）执行——具体执行动作，例如实验场景中的数据输入。

4）终止——终止实验并设置环境和数据采集机制处于非活跃状态。

5）分析——研究获取的数据集，以识别每个被评估候选者所取得成就的等级。

254
~
255

14.3　执行分析

软件分析实践对应用于软件架构的三种权衡研究做了区分：面向需求的研究、面向功能架构的研究和面向物理架构的研究。软件架构的每个元素都有其特性，在执行权衡研究时需要予以考虑。后续章节将对这些方面分别进行阐述。

14.3.1　评估需求候选方案

需要在需求领域执行需求权衡研究，以保证涉众和软件工程团队都能充分理解每个需求的意义。在需求领域执行权衡研究有如下原因：

1）需求是一种沟通机制。用来表达需求的语言可能本来就具有二义性。语言天生具有二义性可以使得它在特定的环境中有不同的含义。对于产品和服务的正式需求，说明交付物的协议必须是无二义的。因此，确定对涉众需求与期望的清晰理解对建立可实现的软件需求基线至关重要。

2）涉众需求和期望中可能存在相互冲突的要求。涉众总是只关注他们自己的需求，因而需要使他们了解个体需求与他人需求产生了怎样的冲突。要解决涉众冲突需求，可以进行实验以在涉众需求中取得平衡。

3）软件产品使得业务流程和操作流程自动化。涉众通常对他们的业务流程非常熟悉，但是软件人员可能并不理解他们使用的相关业务术语。业务模型提供了一种机制，它将业务流程转化成软件操作人员可理解的描述方式。

4）需求可能夸大了性能要求。需求对数据处理事务强加了性能等级要求。有必要建立一种业务模型来评估这些严格的性能要求是否合理。相关软件和计算机技术能否实现性能要求必须经过相关操作和计算分析加以证实。

14.3.2　评估功能候选方案

功能模型是一种将软件需求转化为逻辑表达或行为表达的方式，它关注数据处理事务的执行。通过功能分解帮助进行软件结构配置。有多种方式可以实现数据处理事务以及功能点细分。对功能架构进行权衡研究的最主要原因是：

256

1）评估数据处理事务行为。当条件反馈数量增加时，事务可能变得非常复杂。理解所有可能出现的差异性，通过评估选择最佳方案。

2）评估资源使用率。每个事务都会使用各种比例不同的资源，需要确定如何调整数据处理负担以确保稳定的资源利用。

3）评估数据处理死锁情况及其解决方案。当两个数据处理事务竞争同一个资源时可能造成死锁。了解方案所涉及的资源利用以及分配管理策略有利于避免死锁。

4）评估失效模式及其可能反馈。要了解处于失效时最适当的反馈策略，需要涉及评估方案的行为功能。失效的起因有很多种，每一种失效模式都可能会影响数据处理事务的控制流。需要评估失效条件的不同候选响应方法并确定最佳的响应。

14.3.3　评估结构候选方案

结构配置体现了软件单元如何经过组织形成一个结构化的组件。它涉及与软件产品高层概念结构相关的组装和集成任务。必须对优先的结构方案和集成策略进行评估以理解与可选配置相关的性能和物理特点。对物理架构进行权衡研究的主要原因有：

1）评估结构单元安排是否合理。结构单元的安排通常会对软件功能的表达加以限制。应该对结构单元组织为结构组件的可选方案进行分析以确定最可接受的结构方案。

2）评估软件集成策略。软件集成方法影响软件集成和测试方面的工作量。集成策略同样会影响软件性能以及资源利用率。评估候选的策略模式有助于理解集成策略的结果。

3）评估结构解决方案的完整性。必须对结构解决方案进行评估来确定其与结构设计指
257 导原则和原理的符合程度。若没有遵守指导原则，需要评估该处不符合的地方是否严重影响了操作性能或者结构稳定性。其他结构方案可能能够解决该偏差，但是可能对产品性能和稳定性有不利影响。

14.4　评估项目影响

当候选方案经过技术方面的分析后，需要评估如果将它实施到现有项目中会产生怎样的影响。初步评估涉及一些隐含的软件实现问题、操作问题以及支持问题。然而，关于选择哪个结构设计方法的最终决定必须考虑到项目资源适应设计计划的能力。

14.4.1　评估开发影响

需要对方案进行评估来确定将其实施到当前项目和技术方案中的效果。需要对集成的计划、进度和技术工作包进行审查，来确定候选方案是否符合和完善预期工作任务。软件工程团队需要与开发团队就相关技术规划、进度以及工作包进行协商，目标是确保找到一个架构方案使之能被容纳到当前资源中。

架构方案的元素可能会对开发团队将设计方案纳入其计划的能力提出挑战。每采用一个架构解决方案，就是一个技术风险。当与实现项目目标无紧要关系时，技术风险将会减小。开发因素包含实现、测试以及部署软件产品的能力。相关因素包括项目操作稳定性、可用性、用户培训以及软件支持。因而可能需要一种创新的、非常规的编程范式、语言以及计算平台。在确定采用架构方案之前就需要将软件产品生命周期因素范围设置好。这种集成产品与过程开发（IPPD）理论的核心思想是在评估一个架构方案时考虑整体生命周期中的所有因素。

14.4.2　评估项目影响

258 项目团队应该对与不同候选方案执行相关的开发影响和风险进行评估。需要对项目计划、进度以及工作分解结构进行审核来评估该候选方案是否符合并有助于完成项目目标。我们期望能够找到一个与当前项目资源相符合的架构候选方案。项目团队可以提出应用项目储备资源来适应这种不确定性并预防技术风险。对于将架构解决方案融入一个项目中所引发的风险必须进行评估，来为潜在困难提供适当调整余量。

14.4.3　确定项目执行策略

为了将架构方案融入当前项目，需要为其建立相应的策略。这涉及对必须调整以适应当前架构方案的项目方案、进度及预算进行评估。每一个执行策略都要考虑到将设计方案融合进产品配置中的返工工作，包括调整软件产品规约以及图表来反映方案，以及调整软件开发后期相关概念和文档。

项目方案和进度安排必须具有灵活性以适应技术复杂度提高而产生的偏差。需要密切监控项目风险并准备应急策略以在风险真正出现并威胁到项目目标达成时辅助进行路径纠正。

14.5　评估权衡研究结果

架构候选方案根据权衡研究的成功标准进行排序。一般优先选择那些提供特性较为平衡

的方案。需要确定将实施策略落实到项目和技术梯队的具体行动方案。权衡研究的结果需要进行文档化来保存架构设计决策的历史记录。这些记录为企业提高软件工程实践成熟度提供依据。

14.5.1　为架构候选方案排序

成功概率更大的架构候选方案会被优先考虑。每个权衡分析是不同的，所以不可能为计算成功概率提供一个标准算法。然而，在权衡项目和产品的成功标准时应该考虑表 14-2 中所示的因素。 259

表 14-2　项目成功因素

项目成功因素	说明
及时性	能够在项目截止日期之前完成的概率
成本合理性	能够在规定的资源（员工、设施、设备、资金等）范围内完成的概率
置信水平	规避风险，并能使项目高效、低成本地完成的概率
涉众满意度	对涉众需求的满意程度
架构完整性	是否符合架构指导原则
架构稳定性	产品能够持续演化和改进的健壮性
价值主张	对比投资成本（购买价格、培训以及运营费用）和支持成本（客户支持服务以及产品支持服务），产品所产生的效益

对于排好序的候选方案集合，首先显示的是最佳方案。其余的候选方案从技术角度、程序安全角度以及项目角度评估与当前最佳方案的差异。这些差异应该提供证实优先方案优点的汇总。这个排好序的候选方案集与最佳方案的执行策略被一起提交给涉众委员会。如果涉众对该权衡研究得到的结果和结论有任何质疑也可以在此时提出。

14.5.2　确定优先行动路径

需要对执行策略进行调整以应对所选择的架构设计方案的相关工作。最终行动方案需要描述如何将选择的架构方案融入当前软件架构中。架构决策必须指出明确的方向使得架构能够和设计决策很好地融合。决策体现了对问题空间进行深入调研后的结果，也使得人们对于一个架构解决方案必须包含哪些特性有了更好的了解。执行方案说明了架构解决方案如何被包含进软件的架构、文档以及工作计划的当前状态中。

每个技术团队都需要根据新的工作目标更新他们的技术方案、进度以及工作包。以前的方案、进度和工作包的详细任务栏中用抽象的占位符表示，而现在由于设计方案更新，以前 260
的抽象化占位符的位置可以填写出详细的设计细节。而以前的图纸、图表、规约以及其他设计工件也需要进行改进并加入当前设计方案中。这不仅改进了设计表达，也使得软件架构向更加完备的方向迈进了一步。

14.5.3　将权衡研究的决策文档化

在确定优先行动路径后要将其文档化，这是权衡研究技术团队以及参与制定决策的涉众代表们行动的理论依据。从细微的架构难题，到涉众需要、软件需求、功能架构以及结构配置之间的策略融合，权衡研究在其中的重要性不同。决策文档的等级需要反映出架构重要性的不同。

权衡研究报告需要包含关于方法、候选方案以及结果的以下方面：

1）技术挑战——调研中的工程难题以及该问题对于架构上下文环境的重要性。

2）权衡研究方法——评估架构候选设计方案的方法。

3）候选方案——各个权衡研究领域范围内的待评估的架构候选方案列表。

4）成功标准——成功因素以及它们在选择最佳方案时被赋予的相对权重。

5）分析结果——度量候选方案的效用、开发成本和项目成本以及对进度的影响，并对候选方案进行排名。

6）决策——最终选择的设计方案及其选择原因。

7）执行策略——将设计方案融入当前软件架构中，并更新当前工作（如程序设计、代码单元、集成的软件组件）以及文档。

14.5.4　优化执行策略

制定权衡研究执行策略的目的是使得架构解决方案能够很好地整合到当前架构和适应相关的技术和项目计划、进度和工作包。每个技术小组需要定期汇报在当前工作中加入了更新方案后的进度和成果。如果由于采用了新的技术方案后产生了问题，需要交给软件工程小组进行评估和审议。将解决方案融入当前架构或其他工作制品时可能出现非预期的难题，执行策略需要据此加以调整。

261
/
262

软件验证和确认实践

软件验证和确认（verification & validation，V&V）实践的目的是证实软件架构元素之间的一致性。这意味着确定架构所表述的软件设计已经被恰当地建立以及配置能够满足涉众需求。软件验证和确认实践有时会与测试和质量保证混淆，但是在软件工程学科里，V&V 包含了对软件架构的评估以确定其适合转换到开发的软件实现阶段。软件架构必须进行不断验证以确保其内部一致，并不断确认其满足涉众需求。V&V 必须确保软件设计能够实现预期的目标，从而支持软件项目的进一步投资。图 15-1 说明了软件工程 V&V 技术如何协助通过概要设计评审（PDR）、关键设计评审（CDR）以及测试准备评审（TRR）里程碑。

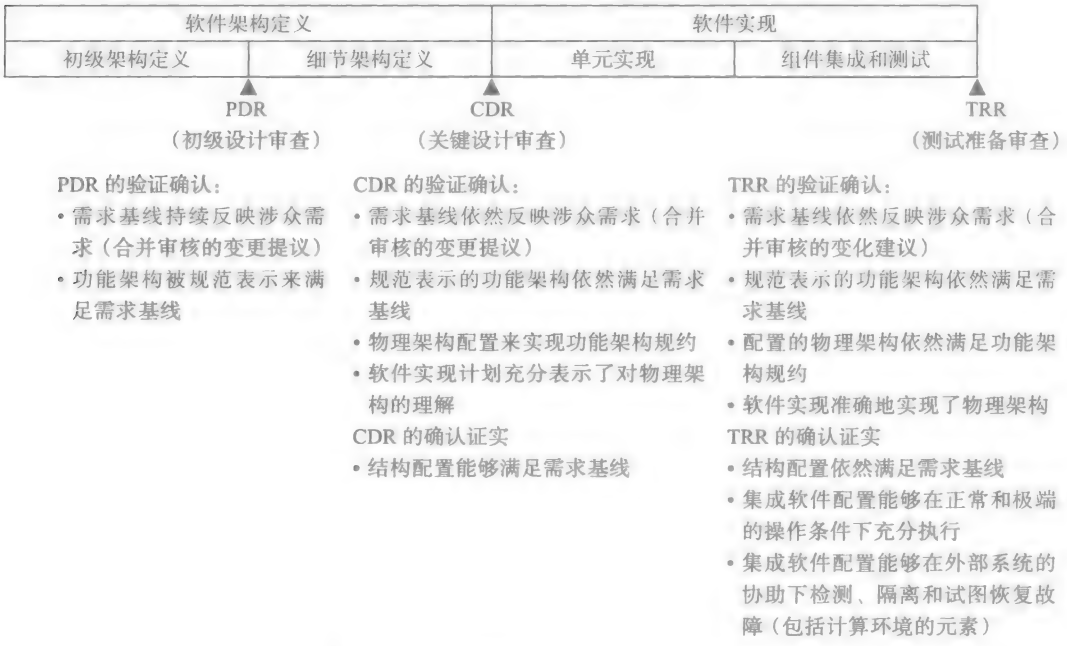


图 15-1　V&V 在阶段审查中的变化

验证确保架构中的每个元素与其构思来源的先前元素一致。在软件需求案例中，验证确保软件基线是从已确认的涉众需求正确导出的。验证提供了对软件架构满足涉众需求和项目约束的充分性的增量式评估。验证还能够确保软件实现计划的遵从，以及物理架构的实施。

确认证实结构配置能够满足它的预期用途，这包括证实完整的结构设计在一般和极端执行条件下能够充分地执行数据处理事务。确认评估物理架构的鲁棒性来满足需求基线、涉众需求和操作的工作负载，以及为预期的产品演化提供一个稳定的框架。这包含评估对终端用户以及外部系统错误（包括与计算环境中的元素相关的错误）的软件响应。

V&V 任务有时会被认为是软件质量保证或测试和评估工作的一部分，然而软件质量保

证关注确保策略和过程符合诸如软件设计和编码之类的标准。测试和评估一般关注软件产品或过程是否满足规约，以及确定产品部署是否准备就绪。因此，本章的任务是在软件工程背景中提出 V&V。图 15-2 给出软件 V&V 任务的概览。

图 15-2　V&V 任务

V&V 任务提供了 V&V 应用到建立详细的、一致的和可实现的软件产品架构的综合范围。尽管有其他软件相关的任务与这些职责重叠，但它们通常作为支持软件开发相关方面的附属工作而存在。V&V 任务的目标是证实软件架构工程指标的有效性。因此本章确定的任务是形成在软件工程环境里讨论 V&V 的基础。

V&V 任务被制定来确保软件架构是完整的并且内部一致的，其重点是确保软件架构的结构完整性、持久性和简易性。V&V 任务被分为 4 个主要活动，如下所示：

1）定义 V&V 策略；

2）验证软件架构；

3）确认物理架构；

4）文档化 V&V 的结果。

15.1　定义 V&V 策略

必须将 V&V 活动和任务整合到软件工程计划中，旨在评审已完成的工作产品并确保它们的充分性。这涉及对于每个架构设计机制必须满足的业务场景的相关假设的检查。V&V 任务应当由那些拥有评估设计机制技术特点的相关技能和知识的软件工程团队成员来进行。

15.1.1　建立 V&V 范围

应该对软件工程工作产品进行审查以识别 V&V 机会。应该建立能够按照其对软件架构贡献来查询工作产品的任务。软件工程计划应当纳入 V&V 任务，并应用与每个工作产品的重要性相符的资源。表 15-1 列出了一系列在开发 V&V 活动范围时应该考虑的 V&V 任务元素。

265
|
266

表 15-1　软件 V&V 任务元素

实践	描述	重要性
需求基线验证任务		
验证软件需求规约	确保每个软件需求能够跟踪到正当的涉众需求或期望。	必要（强制）
	确保每个需求都是唯一的、可量化的和可测试的。	
	确保一组需求是集体一致的和不重复的。	
	确保软件需求以与需求权衡研究记录相一致的方法来陈述	
验证软件接口规约	确保每个外部接口按照有益于数据处理事务的方式来规范化。	必要（强制）
	确保接口提供必要的数据或信息来支持数据处理事务。	
	确保接口并没有降低或干扰效率或有效的数据处理事务	
验证业务模型	确保业务模型精确地展示了应该描述的操作或业务流。	必要（强制）
	确保每一个业务模型的元素被恰当的定义，来防止错误的解释。	
	确保操作行为建立了关于有效性和性能的软件度量	
验证需求背景	确保每个需求都被关键涉众认可，包括工程管理者和软件工程团队。	重要（可选）
	确保需求的来源在需求追踪机制中被正确地文档化	
	确保每个需求都被业务模型实体化	
功能架构验证任务		
验证功能分解	确保软件需求在功能上的分解与功能权衡研究记录相一致。	重要（可选）
	确保已经评估功能复杂性，并且功能分解与权衡研究复杂度解决方案相一致	
验证行为模型	确保功能行为模型与高层业务模型相一致。	必要（强制）
	确保外部接口与接口规约相一致。	
	确保资源的可用性与计算环境规约相一致。	
	确保功能时间轴与操作时间轴相一致	
验证功能规约	确保功能单元和组件规约正确的包含来满足高层规约。	必要（强制）
	确保功能单元和组件规约恰当地反映出从行为模型中得到的性能特征	
验证整个功能架构的需求可追踪性	确保需求的可追踪性机制是当前的，并且提供功能架构元素中的可追踪性	重要（可选）
验证到需求基线的需求可追踪性	确保需求的可追踪性机制是当前的，并且提供从功能架构中顶层元素到需求基线的可追踪性	必要（强制）
物理架构验证任务		
验证结构单元规约	确保结构单元规约正确地包含功能单元规约特征	必要（强制）
验证概念设计	确保概念设计的元素连贯地展示显性数据处理事务	重要（可选）
验证软件集成策略	确保软件集成策略巧妙地弥合软件设计鸿沟。	必要（强制）
	确保软件集成测试正确地测试了完整的结构组件	
验证结构组件规约	确保每个结构组件规约建立了关于完整结构元素和设计机制（例如用图表示的用户接口元素）的混合行为	必要（强制）
验证整个物理架构的需求可追踪性	确保需求的可追踪性机制是当前的，并且提供物理架构元素之间的可追踪性	重要（可选）
验证到功能架构的可追踪性	确保需求的可追踪性机制是当前的，并且提供功能单元和结构单元之间的可追踪性	必要（强制）
软件实现验证任务		
验证软件实现计划——结构单元的实现	确保软件实现计划充分考虑了设计、编码和测试每个结构单元所必要的工作量	额外信息（有益）
验证软件实现计划——集成策略	确保软件实现计划充分考虑了组装、集成和测试每个结构组件所必要的工作量	额外信息（有益）

（续）

实践	描述	重要性
验证软件单元的一致性	确保每个软件单元的实现符合结构规约	必要（强制）
验证软件组件的一致性	确保每个软件组件的实现符合结构规约	必要（强制）
配置确认任务		
确认具体数据处理事务的完成	确保物理架构准确促进具体数据处理事务	必要（强制）
	确保物理架构包含处理适当和不适当的全范围用户输入的设计机制	
	确保物理架构包含处理适当和不适当的全范围的外部系统接口的设计机制	
确认具体性能需求的完成	确保物理架构充分配置来完成规范的性能需求。	必要（强制）
	确保物理架构充分配置来稳定资源利用率需求	
确认从注入故障条件的恢复	确保物理架构充分配置来从预期的错误或退化的业务模式（计算环境或外部系统）中检测、隔离和恢复（回到前一个健康的状态）	重要（可选）
确认在高工作负载下的执行	确保物理架构配置充分能在高工作负载期间执行数据处理事务	重要（可选）
确认计算环境的充分性	确保计算环境规约充分能保证软件产品的操作性能	必要（强制）

V&V 任务应当由一个软件工程团队成员所领导的高级软件人员团队来执行。这个 V&V 团队必须配备知识渊博的、有经验的成员，因为他们承担着确保主要架构产品的完整性和准确性的责任。

15.1.2　建立 V&V 方法

对于每个 V&V 任务，必须确定进行评估的方法。验证旨在建立架构设计的精确度和完整性，而确认旨在确保软件产品展示了正确的操作事务、特征和质量特点。以下是一些能够应用到 V&V 任务的技术，被分成 4 类 V&V 方法：

1）文档评估。关于架构图表、图示和规约的技术评估以达到 V&V 目标。

2）同行评审。通过专家同事对架构元素健康度的技术评估来达到 V&V 目标。

3）静态分析。应用关于软件行为、性能和资源利用率等的数学或科学预估来评估 V&V 目标的实现情况。

4）动态分析。使用计算统计方法来量化性能、资源利用率和数据处理效率的预期水平来评估 V&V 目标的实现情况。

267
~
270

15.1.3　建立 V&V 过程

执行 V&V 任务的过程必须要准备好建立一个预期方法来完成一系列行为。该过程提供关于任务执行方式、调查项目和调查结果报告方式的详细指令。V&V 过程必须清晰地标明待验证的项目、与程序相对应的工程任务和工作包，以及 V&V 报告的接收者。

V&V 过程必须确定验证任务所需要的材料，必须确定负责提供架构工件以供评估的组织。每个 V&V 任务的计划日期不应该在产品被安排完成之前确定。验证任务能够在未完成的产品上执行，但这是临时的或初步的验证工作。在被转换到开发的下个阶段前，验证完整的架构元素是必要的。因此，技术和工程计划必须说明 V&V 任务，并且在进行技术审查前必须执行正确的行为。

15.2　验证软件架构

软件工程验证任务致力于确认软件产品架构是完整的、内部一致的并准备好过渡到软件开发的实现阶段。这包括验证软件架构的元素（需求基线、功能和物理架构），以及确保软件实现遵从结构配置规约。

15.2.1　验证需求基线

必须对需求基线进行验证以确保每个需求都能够跟踪到涉众需求并且基线表示了软件产品适当需求的完整集合。需求基线由软件需求、接口规约以及导出它们的业务模型组成。

15.2.2　验证功能架构

必须对功能架构进行验证以确保功能分解是不过度复杂的，并且性能指标在子功能之间有效分配。顶层功能必须可追踪至软件需求基线和业务模型。必须对行为模型进行验证以确保其准确表达数据处理事务和控制场景。必须对功能规约进行验证以确保其准确遵从功能分解并表达所分配的性能指标。

15.2.3　验证物理架构

必须对物理架构进行验证，以确保结构配置为软件实现、集成和测试提供了一个不过度 271
复杂的框架。必须对结构单元规约进行验证，以确保其充分整合了所分配的功能单元规约并提供到原始功能单元规约的可追踪性。必须对概念组件进行验证，以确保其正确反映了主要的数据处理事务。必须对结构组件规约进行验证，以确保其具备从结构配置低层元素整合所得的功能和性能综合特征。结构配置中的需求可追踪性也必须进行确认。

15.2.4　验证软件实现

验证软件实现计划、进度和工作包符合架构配置。这包括设计、编码和测试架构单元以及执行软件集成和测试的工作。必须对结构装配工作包进行审查以确保为软件测试桩的准备和验证分配了足够的资源。软件单元设计同行评审应当提供验证其遵从结构单元规约的证据。组件集成同行评审应当提供验证其遵从结构单元规约的证据。

15.3　确认物理架构

软件工程确认任务致力于确认物理架构是完整的，能够满足软件需求基线并准备好过渡至软件开发的实现阶段。

15.3.1　确认结构配置

必须对结构配置进行确认以保证其满足规定的性能指标。这包括工程分析和关于数据处理事务时间间隔和资源利用率等数学测算。另外，结构配置的性能必须建立那些决定计算环境和外部系统和应用接口性能的预期操作性能度量。这些决策必须保证结构配置能够满足需求基线所规定的性能指标。

15.3.2　确认集成软件配置

必须对集成软件配置进行确认以保证其满足规定的性能指标。软件配置可能包括两个或

272 更多的软件配置项目，必须确认它们能作为一个集成产品有效并且高效的运行。这涉及对于集成软件配置的工程分析，其中包括对特定计算环境以及外部系统和应用接口的性能的考虑。集成软件配置的确认必须为一般、严重和过度条件下的数据处理工作负载建立预期操作性能度量。在决策中必须建立引起软件配置开始退化和反应迟钝的数据处理工作负载。该确认可能需要利用软件集成、测试记录和计算环境基准来推断性能指标。

软件配置检测和响应硬件故障的能力必须通过工程分析验证或符合利用软件整合和测试记录证实。软件配置在特定的降级模式——故障安全、故障可靠、故障弱化中持续运行的能力必须证实。

15.4　记录 V&V 结果

V&V 结果必须进行报告。软件设计缺点必须被识别和隔离到违规架构配置元素中。必须对这些缺陷进行分析以建立一个纠错行为计划。在 V&V 报告中每个 V&V 任务都必须进行记录和归纳。每个 V&V 任务的报告应该提供充分的细节信息来协助修改有缺陷的设计元素。V&V 任务最少应该包含以下元素：

- 识别评估的架构元素；
- 识别参与元素之间的关系；
- 执行 V&V 任务的方法；
- 调查结果；
- V&V 参与者的建议。

每个 V&V 任务的最终报告应当被唯一标识，并输入至日志以记录 V&V 任务标识、标题、任务执行时间，以及归纳调查结果的报告的标识和日期。V&V 调查结果必须发布，并且分发给受结果影响的软件组织。调查结果必须清晰地识别软件架构中有缺陷的元素、发现的缺陷以及能够解决缺陷的推荐执行过程。修改有缺陷的设计元素的执行过程必须与软件开发工作的状态、缺陷的严重性和在当前项目进度表中解决缺陷的能力相一致。那些不能在软件产品发布前解决的缺陷必须用应急方案避免来防止用户无意碰到。只有放弃在软件发布时未能满足的需求被批准后才能认可应急方案。

273 / 275

软件控制实践

软件控制实践涉及一系列旨在为软件工程活动、产品配置和变更过程提供稳定性的任务。这些任务代表了配置管理和项目控制任务的技术版本。其中包括评估某个变更请求或提议对软件架构、技术方案和架构设计工件的潜在影响。

软件控制任务必须要记录相应的配置记录，从而保证软件架构元素、设计决策、涉众需求的变更请求和提议的可追踪性（图 16-1）。项目层面的配置控制通常实施于软件配置项层面的软件产品配置。软件控制提供了对软件架构演化过程中配置管理的监督。总而言之，软件控制任务为软件架构演化、权衡研究、风险以及吸纳变更请求和提议提供了更详细的控制管理。软件架构的演化具有连续性，直到将软件结构配置提交到项目层面的配置控制时才停止。这一提交行为也意味着当前的软件架构已经变得相对稳定，能够适应外部因素的改变，并已经对开发项目整体结构进行了详细的、规范化的描述。

275

图 16-1　软件控制任务

典型的项目级配置管理（CM）实践适用于软件产品配置及其相关配置项。技术配置控制是对整个软件架构进行监督，从而对软件需求在功能和物理架构中的分配和可追踪性进行管理。这部分工作内容包括识别和规范结构单元、组件以及内部接口。

当有变更请求和提议时，必须要区分影响项目或合同范围的变更与只是影响当前技术工

276

作的变更。影响项目或合同范围的变更请求和提议也许包含影响技术工作这方面，然而这样的变更对项目的影响比技术或工程影响更具本质性。因此，在项目配置管理系统中必须要对这类变更进行正确处理。对于只是影响技术工作的变更请求或提议，应当将它们归类到技术配置控制系统中进行处理。

16.1　配置管理

软件配置管理为软件架构演化过程中的配置提供日常管理，它包括用项目唯一标识符识别软件架构的每个元素，并维护每个元素的配置状态记录。

16.1.1　识别架构元素

软件架构中的每个元素必须具有唯一标识，这些标识用于关联软件架构元素的当前和历史信息。技术配置识别可应用于软件架构定义期间出现的架构元素。在定义的早期，架构配置会随着工程分析对方案的改进而进行反复的迭代变更。在定义期间，需要小心谨慎地分配配置标识符直到设计变得稳定。

架构元素的识别涉及以下几点：1）软件需求；2）功能设计层次，行为模型，功能规约；3）结构层次，工程组件，集成策略，模型，原型和结构规约。此外，架构配置识别必须确立产品版本和基准从而表示架构配置的演化。架构版本管理实践为整合单个架构分区的设计变更、修改以及变化提供共同的参考点。[⊖]当三个架构分区被合并为一个统一的软件架构时，架构基准管理实践为它们提供了参考。

架构元素的识别可以为以下工作提供帮助：

277

- 追踪架构元素到赋予其特征的设计工件；
- 展现整个软件产品生命周期中软件架构的一致表示；
- 建立和维护软件架构中每个元素的工程记录；
- 提供建议和授权变更状态，便于架构元素定义和架构文档更改的实施。

16.1.2　维护架构状态

要维护软件架构的状态，从而提供架构准备过渡到软件开发或部署下一个阶段的就绪进展指示器。软件架构作为统一的产品描述，其包含的三个分区可能具有不同的成熟度水平。当软件架构的三个分区均变得稳定和统一后，它们的配置应该置于配置控制中，此时软件架构的定义也被认为是完备的。

架构分区中每个软件架构的元素都应该被监控，从而明确元素当前状态。元素状态的分类建议如下：

1）起草——元素已经被认为是分区中的一个必要条目，但是元素的生命周期还没有确定。

2）原始——元素已经被确定为分区中一个基本组成元素。

3）替换——元素可以替代一个起草的元素，或者在设计解决方案竞争中被认可。

4）控制——元素已经拥有完全定义并指定为分区的一部分，置于技术配置控制中。

⊖　本文所述的架构分区是指软件架构描述中的一个子部分。每个分区包括以下子部件中的一个：1）需求基线；2）功能架构；3）物理架构。划分表示对组成部分进行划分。它可以对一个架构的分区（单个分区边界内的所有元素）进行划分，或者对软件架构的分区进行划分。

5）过期——元素已经被分区排除，但是仍然保留在元素库中，并提供历史依据。起草的架构元素作为原始的或者替代的设计概念，最后被认定为非必须时，那么可能从状态记录中移除，但是这个元素仍然保留在库中为以后的开发提供参考。

软件架构状态记录的维护是一项重要的工作，可以支持进展汇报，追踪决策的改变对软件架构文档、模型、设计工件造成的影响，保证设计决策能够正确集成到架构中。重点必须要放在组成架构的元素上，而不是架构文档、模型和工件上。需求、功能、结构单元和组件、数据、条目等表示架构不同层次的配置元素，是软件产品的组成成分。因此有必要了解架构元素定义的状态，从而获得整体开发工作的状态。 278

16.2　处理工程变更包

需要正确处理工程变更请求和提议，从而确保涉众或软件开发团队成员提出的改进能够被正确考虑。工程变更请求表示对软件架构在现有项目范围内所建议进行的修改。这意味着该变更不会影响当前项目的工作量和进展。变更提议表示处于当前开发工作范围之外并需要对项目资金和 / 或进度表中的截止日期进行调整的修改。为澄清涉众需求或修正错误假设而提出的修改不应看作是需要变更控制的修改。

16.2.1　记录工程变更请求和提议

每个变更请求或提议应该记录到一个变更追踪日志中或存储于库中。这些请求修改软件架构的记录，为软件架构稳定性提供度量的依据。

架构分区中的修改数量应当随着架构的演化和成熟变得越来越少。理想情况下，在对分区进行技术评审时，针对架构分区的修改请求就已经停止。

工程变更记录应当包含以下信息，从而保证可追踪性以及状态报告：

- 变更请求或提议的数量；
- 变更开始日期；
- 变更的发起者；
- 变更的类型；
- 变更的优先级；
- 变更对主要架构分区的影响（需求基线、功能或物理架构）；
- 变更的处理方式；
- 处理的日期；
- 同化状态。

16.2.2　准备变更评估包

需要装配变更评估包用于为评估所请求的架构修改提供一致的基础。变更评估包应当由受变更影响的基本架构工件组成。变更评估包中应当包含足够的信息、文档和架构工件，从而帮助评估者评价这个变更请求是否可以用于软件架构、技术和项目计划。因此，变更评估包应该包括表 16-1 所示的内容。 279

变更评估团队成员应当对变更评估包进行审议。变更控制管理员应该确认变更评估团队每个成员对这个变更评估包的接受度，保证在他们的工作任务和预算限制范围内进行变更评估工作。

表 16-1　变更评估包的建议内容

内容	描述
变更的起源和追踪	需要提供如下信息： • 变更的起源 • 变更的标识符 • 发起变更的实体或请求的发出者 • 将请求日期记录到变更追踪系统
变更的描述	为架构修改请求提供一个警示性描述，包括： • 受变更请求影响的架构元素和分区 • 变更对当前架构元素的哪些地方造成影响 • 架构修改所能产生的预期效益（如操作的改进，客户或消费者的认同，扩大市场渗透率，易用性，减少使用该软件所需的培训和教育成本）
与评估相关的架构工件	标识出作为包附件的架构图表、文档等或者它们在数据库中的存储位置。还应该提供定位和存取工件的说明
评估说明	提供说明让评估者了解如何对变更请求进行评估，并且指出变更所需关注的范围。应当重点关注如下几点： • 变更的技术优点 • 变更后会对架构设计造成哪些影响（例如架构复杂性） • 是否对技术方案和进度造成影响，如果有则明确有哪些影响 • 是否对技术工作包造成影响，如返工、工件修改、对软件实现和测试的影响
评估里程碑	评估过程中有如下里程碑式的阶段： • 对变更请求的各项评估进行综合，获得一个统一的决策 • 软件工程集成产品团队（SWE-IPT）的决策 • 软件变更控制委员会（CCB）的决策

280

16.3　变更评估

变更评估团队成员应该对提出的架构修改进行评估，明确采用这种变更的优点和后果。评估团队必须证实变更的内在价值，包括如下几点：

- 所提议变更的技术优点（在功能、性能、可用性、可支持性、操作方面的改善）；
- 对软件架构采用该变更的结果预测（复杂度的增长，改变的范围（深度和宽度），变更对架构的其他元素造成的影响）；
- 变更对技术方案和进度的影响；
- 变更对技术工作包和资源分配的影响；
- 变更提议伴随的其他资源的变化；
- 每个技术团队对变更的接受程度，包括工作负载、预算约束和工作包的再分配三个方面。

考虑当前项目结构中资源的使用，变更提议对软件开发工作的完成也许是一个负担。为弥补架构设计特性的重新构造和调整而提出的变更可能需要附加资源。必须确定同化变更所需的附加资源足够抵消所增加与将变更融入软件开发日程相关的工作量。

16.3.1　评估变更技术优点

必须要对变更包进行评估，从而确定采用这种变更所带来的技术优点。应当给出变更的商业案例从而展示变更的预期效益。提出的变更给软件架构带来的技术优点需要明确给出，

包括对性能、可重用性、可支持性所带来的提升。技术优点的识别首先需要明确软件架构变更的优点和缺点。这些技术优点用于指导变更，让变更有助于实现架构的指导方针和性能要求。

技术优点指的是那些不能直接度量或者科学证明的属性。它是评估团队成员针对提出的变更进行充分的调查和公开讨论所得出的观点。对变更的技术优点进行评估目的是明确所提出的变更是否能够提高软件产品的操作性能和架构完整性。评估针对软件架构所提出变更的 281
技术优点时应当考虑下列影响：

1）变更将如何影响数据处理事务的效率？

2）变更将如何影响结构稳定性（业务或者计算环境对变更的抗干扰能力）？

3）变更将如何影响产品的可扩展性（适应业务环境中加载变更的能力）？

4）变更如何影响计算资源的有效利用率？

5）变更如何影响软件架构的复杂性？

16.3.2　评估架构影响

每个提出的变更将会对已有架构框架引入新的设计元素或者机制，从而影响架构的稳定性。与开发过程中提出的变更相比，软件工程早期提出的变更可能不会付出太大的代价。确定实施该变更后可能对架构造成的影响，包括理解变更对架构元素的波及效应，必须要对每个与变更关联或紧密耦合的架构元素进行识别。为了理解所提出变更的深远影响，需要关注可能受变更干扰的元素。

评估架构变更的技术优点应该考虑以下问题：

1）变更在软件架构中影响有多广？

2）架构变更的总量是多少？

3）变更对架构造成什么分歧？

16.3.3　评估技术工作包影响

必须重新评估目前完成软件开发项目的技术工作，应考虑到将变更融入软件架构及其工件的工作量。它涉及变更引起的已有文档、模型、图表的返工，包括以下几方面：

- 更新规约、图表、绘图、文档引用、模型；
- 修改软件实现工作的任务和规划，包括必要的代码返工，软件单元组构件的测试； 282
- 修改软件测试和评估工作的分配、规划及过程；
- 软件后期开发过程中对变更的适应。

必须要对软件开发工作中预期的返工和更改进行评估以准确了解技术工作包变更的影响量级。当调整工作包时，需要考虑工作间的依赖关系。对工作包变更导致的随后任何工作的延期进行识别是很重要的。工作所需资源预算必须要重新评估，确保它们有足够的供应以支持剩余的技术工作。这种评估必须考虑完成软件开发所剩下的工作以及有助于完成开发的资源（资金、人力、设备等）。

16.3.4　评估技术方案影响

应当将修订的技术工作包假设地纳入技术方案和进度安排，从而明确实现技术目标的任何潜在的影响。改进工作包时产生的影响需要与集成总体方案和进度安排进行综合，从而确

定可能会出现的冲突、缺陷或效率低下等问题。应当对各种技术方案进行重新检查，以确定是否可以用工作量和资源进行弥补，从而设计一个正确的执行计划。

16.4　变更同化

变更同化涉及负责确保批准变更正确进行并融入架构的软件控制任务。当技术或项目层面的 CCB 主席采纳提出的变更请求或变更提议时，意味着变更同化的正式开始。在 CCB 召开会议批准变更前应当先准备好变更通知包。

如果业务案例、技术优点和结果证明当前提议的变更是有利的以及可实现的，那么评估团队应当推动这种变更的落实。变更请求或提议的最终批准由技术或架构层面的 CCB 掌控。当软件架构由技术 CCB 控制时，首席软件工程师或技术 CCB 的主席可以批准将要进行的变更。然而，如果提议的变更需要额外的项目资源支撑，那么只有项目 CCB 的主席或项目经理能够批准当前提议。

16.4.1　发布变更通知包

283

必须要准备变更通知包，从而识别当前采纳的变更所影响的架构元素和设计材料。概要软件设计解决方案应当随着变更请求一起提交。实际的软件设计解决方案必须提前准备，并整合到架构的定义、模型、图表、规约、文档和工件库中。变更通知包应当确定每个变更项的负责组织以及变更完成的日期。完整的变更通知包必须予以发布并传达给负责组织以启动变更的同化。

16.4.2　审核架构变更进展

必须定期查询参与变更包同化的软件组织，从而了解每个变更同化工作的进展。架构变更审核必须要追踪软件架构、规约、图表、绘图、文档、技术方案和模型的变更状态。变更同化的状态直到所有受影响的工件和技术方案都反映了修改后才认为是完整的。

16.4.3　评估项目现状

变更同化状态必须在整个项目现状基础上进行评估。由于涉及变更包同化的软件工程任务的环境多变，因此相关任务状态需要特别关注。当所有关于当前变更同化的任务完成时，应当实施一个将变更集成到软件架构的最终审核。当最终审核通过后，该变更可以被归类为已满足，并转移到变更历史库中。

架构版本控制应识别软件架构及其工件的每个工程版本所涉及的变更请求和提议集合。这意味着必须在技术版本控制下进行软件架构定义的演化。所有工程工件必须与架构版本保持一致，这使得所有软件规约、图表、绘图、文档和模型能够与一个或多个软件产品架构的版本相关联。软件工程任务、软件定义及其工件的同步应该是一个基本的软件工程实践。必须理解软件架构的工程演化，从而对推动每个软件产品版本的变更进行说明。

16.5　软件库控制

在软件工程工作的执行中，获取和存储软件架构定义演化的历史信息有一个先决条件，
284 那就是如何维护库。主要的库在下面的章节中介绍。

16.5.1　维护工程工件库

工程工件库旨在大范围采集软件工程工作中产生的工程工件。这包括为需要的分析、设计和文档工件建立库分区和文件夹。一旦一个工件完成了，它应当被放入一个关联每个软件架构版本的控制文件。在一个架构元素文件夹下建立一个子文件夹，用于采集权衡研究材料、报告和决策依据。

16.5.2　维护变更历史库

软件变更历史库旨在采集软件工程团队处理的每个变更请求和提议的历史信息。它应当存储变更请求或提议的内容、变更评估材料、CCB 确定的结果、变更通知包以及变更同化审核。

16.5.3　维护技术风险库

技术风险库旨在采集软件工程团队确认的每一个风险，包括软件架构定义、计算环境定义、软件实现、软件测试和评估、开发后的过程相关的风险。风险库应当采集初始风险识别备忘录、评估报告、消除策略和风险监测报告。

285
~
286

第三部分

Software Engineering: Architecture-Driven Software Development

软件工程应用的阶段

本部分讨论软件工程和软件生命周期每个阶段应完成的相关组织任务。这些任务都是由参与软件工作的功能组织安排的，包括软件工程、计算环境准备、软件实现、开发后的过程准备和软件测试与评估。

图 1 展示了软件生命周期的一种表示法。这些生命周期阶段用于描述软件产品应如何进行开发、销售和支持。软件生命周期从识别软件产品和计算环境的需求开始。然后在架构定义阶段对产品进行设计，为功能和物理单元与组件开发规约。所有的物理单元与组件都被规约后，产品和计算环境实现就可以开始了。一旦软件单元和组件被集成为完整的产品配置，
287 就可以进行产品验收测试了。成功完成产品验收测试后，就可开始开发后期阶段，包括软件销售、培训和维护。开发后期阶段一直持续到软件产品不再需要维护为止。这些生命周期过程需要进行并行开发，以确保它们为软件销售和支持做好准备。

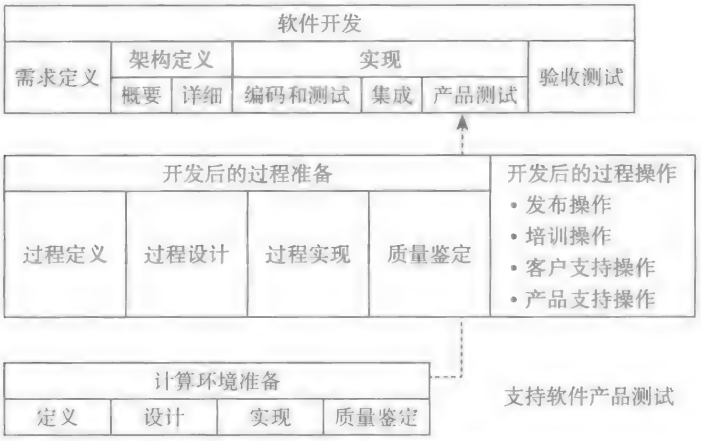


图 1　软件生命周期的阶段

应在软件生命周期的每个阶段执行所需的软件工程过程从而将软件从概念演化为完整并经过测试的产品。软件工程过程为处理复杂性、理解与不同可选设计方案相关的风险直到导出可在程序成本和计划目标内实现的设计方法提供了基础。在软件开发、实现或支持过程中对产品进行设计时都需要使用软件工程过程。软件工程过程为理解可选设计的概念、风险及其对程序目标的潜在影响提供了一个条理化的方法。这有助于软件工程团队选择最优的架构设计配置。

在集成化产品和过程开发过程中可使用图 2 所示的软件项目层次结构。它阐述了在集成化产品和过程背景下的组织角色和职责。

软件：由软件产品、操作计算环境定义和相关开发后期软件支持过程所构成的集成系统。

产品：为满足涉众需求和期望所定义、设计、实现和测试的软件。

计算环境：为产品提供操作环境的计算硬件（主机、服务器、工作站、台式计算系统等）、操作系统、数据库管理系统以及其他产品、通信通道和网络。

开发后的过程：向客户或消费者销售软件产品、提供培训资源和材料以及在软件生命周期中对其进行支持和演化所需建立的过程。

288 复制：为进行数字销售，对软件产品和用户文档进行复制的过程。复制还涉及对最终数字和打印资料进行包装以销售给销售商、客户或消费者。

销售：包装后的产品从来源到分发者、零售商店、客户或消费者的传递过程。

培训：将如何运作软件产品以支持业务过程、人员职责和教育或娱乐目标的知识传递给最终用户的过程。

维护：在产品生命周期的开发后的过程阶段，处理产品缺陷和改善产品的过程。软件维护是对软件问题报告进行记录、调查和处理，并对修复结果进行分发的过程。软件维护可分成两个主要活动进行讨论：客户支持和软件支持。客户支持包括提供在线或电话支持所需的操作。它还包括在需要时为客户提供专业软件产品安装、配置或问题报告方面的帮助。软件支持包括所报告问题的处理和软件结构配置中问题源的隔离。这有助于问题的修正和补丁的发布从而修订可执行文件。如果补丁不能在可接收的时间内进行发布，那么就应该修改操作变通方法并提供给遇到该问题的客户。对软件产品的改进可能涉及一系列开发该软件产品的新版本的辅助项目。

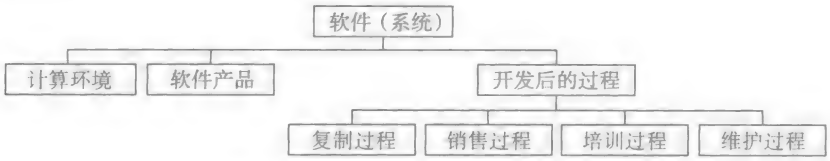


图 2　软件集成产品和过程层次结构

接下来的 4 章阐述了与软件生命周期各阶段相关的任务。它们识别了需生产的产品及其需完成的任务。另外，它们还描述了在临近每个阶段结尾时需进行的技术评审。这些章节还识别和描述了与各软件组织相关的任务。各软件组织都参与到软件工程集成产品团队中，从而为产品生产带来其独特的视角。讨论涉及的软件组织包括：

- 软件工程集成产品团队。
- 软件实现。
- 计算环境准备。
- 开发后的过程准备。
- 软件测试与评估。

289
~
290

第 17 章

Software Engineering: Architecture-Driven Software Development

软件需求定义

软件开发过程中需求定义阶段的目标是将项目涉众的需求、期望和约束转化为一个平衡且可完成的软件需求集合。这包含了软件产品、计算环境和开发后的过程规约的开发。这个阶段的一个主要工件是其计划实现的业务过程或嵌入式系统过程的模型。业务模型为软件产品、计算环境和软件接口需求的获取奠定了基础。计算环境会限制软件产品如何操作来支持数据事务过程。软件需求定义阶段的重点是理解软件产品必须实现的所有数据处理功能。

业务模型为在各种情况和工作量负载下评估软件的整体性能提供了基础。计算硬件、工作站、网络以及相关的技术都有助于提高软件解决方案的性能。计算环境不应过早选定，应等到操作需求分析过后才能选择。操作环境的分析可以确定集成软件的性能需求，其中性能需求又受计算环境中计算设备的约束。软件产品可以通过设计使其更高效地执行于特定的计算设备。经过软件开发这个阶段进行的分析，应该能得到一个架构决策，能为计算环境和软件产品分配需求。

除非选择增量式的或演化的开发策略，否则软件需求定义阶段在软件开发工作中只出现一次。这个阶段为将开发的计算环境和软件产品生成需求规约。为了解决与软件产品或开发后的过程需求相关联的挑战或风险，将在整个软件工程活动期间组织软件需求分析实践。

在软件开发阶段期间，重点是要确保为软件产品和开发后的过程制定的需求是完整的，在项目目标和约束下是可以实现的，并且给项目带来的风险最小。为了对软件产品需求进行分解并将它们分配到软件配置项和计算环境的要素中，需要开展功能分析和分配。在软件开发的这个阶段，软件产品的物理架构应该可以确定软件配置项和计算环境的要素。这样就得到了软件配置项、软件外部接口和计算环境的规约。

为软件实现、计算环境实现、开发后期以及测试和评估团队配备的应该是能够胜任软件工程工作的人员。过度地为这些组织配备人员是没有任何好处的，除非软件架构的状态保证人员的增加能够使得工作效率提升。

为了针对开发的产品进行需求控制，必须制定软件需求基线。这是一个技术基线，将由软件工程集成产品团队进行管理，直到确立正式的功能需求基线。只有已经验证软件架构对所有涉众是明确的、一致的且经得起考验的，才能制定软件需求基线。一旦确定了软件需求基线，任何对该基线的修改必须通过正式的变更控制委员会批准才能实施。一旦确定了功能需求基线，任何对该基线的修改必须通过项目级别的变更控制委员会批准才能实施。

每当对需求提出变更时，SWE-IPT 应该评估变更对需求基线带来的影响。为了针对所提变更确定可选的需求表达，需要执行软件需求分析任务。还需执行软件分析实践来确定最好的需求表达以明确提出。

17.1 软件需求定义的产品

在软件开发的这个阶段，必须生成下列产品：

1）业务模型。业务模型用于描述如何利用产品来帮助业务过程或者嵌入式系统过程。业务模型应将软件产品看作是一个“黑盒”，不应试图去确定产品设计或内在工作的任何方 292
面。业务模型应该描述软件产品如何与操作人员和计算环境的要素配合工作来完成它的预期目标。业务模型应该处理功能序列（包括并发的数据处理事务），每个任务的持续时间，以及数据（接口）与计算环境要素，其他系统或应用和操作人员的交互。每个功能所必需的资源应该被确定并分析这些资源是否可临时获取、可占用或可共享。业务模型必须关注建立测试线程所需的数据处理层次，测试线程将用于确定产品可接受性。

2）初始软件行为模型。软件行为模型用于描述软件产品如何与操作人员和计算环境配合工作来完成数据处理事务。行为模型是业务模型的分解，它的重点在于理解分配到软件产品的操作功能将如何执行。软件行为描述了与每个事务的完成相关的功能流、控制流和数据流。必要时，行为模型应该处理每个数据处理事务以便在软件配置项或产品级别上充分地详细描述软件行为。该模型描述的是必须执行什么数据处理功能，而并非描述它在物理上或结构上如何完成。

3）软件需求规约（software requirement specification, SRS）。已分配到软件产品的功能需求和性能需求必须在 SRS 中进行文档化。这些需求可能分配到多个软件配置项规约中。它应该捕获功能行为、数据交互需求和接口需求，包括业务模型确定的人－机接口需求。这些接口需求可以在各自的软件接口规约（见第 5 项）中文档化。软件销售、培训和维护需求也必须详细说明。这些开发后期的需求可以在各自开发后期的概念文档（见第 7 项）中进行文档化。软件需求规约应该制定一个能确定软件鉴定方法（分析、演示、检验或推断）的矩阵，其中软件鉴定方法可用来决定一个测试的成功与否。

4）计算环境需求规约。已分配到计算环境的操作需求应该在计算环境需求规约中进行文档化。它应该确定计算处理的资源需求，包括交互和网络带宽、最小可接受的并发处理事务的数量以及其他事务质量的度量[⊖]，例如，互用性、可移植性、可量测性、安全性、可维 293
护性、复杂度和吞吐量。如果期望软件产品可在多个计算平台上操作，则需要准备多个计算环境规约以说明每个平台的不同特性。

5）软件接口需求规约。业务数据交换的需求应文档化以处理所有的接口，包括人－机接口。软件接口需求规约应确定产品、其他系统、应用和计算环境要素中的每个接口。每个接口需求必须根据交互的信息内容，以及参与的配置项之间数据传递的方法进行详细说明。

6）软件测试和评估计划。软件测试和评估计划应准备好以解决软件验收测试策略以及如何和何时安排软件质量保证检验。当应用测试到 SRS 中规定的每个需求时，软件测试策略应解决软件鉴定方法。软件质量保证检验应该确保软件开发任务是根据已制定好的过程进行的，以及产品需求、架构和实现都趋向一个完整且一致的产品解决方案。

7）软件开发后的过程概念文档。应准备好关于产品如何复用、销售、维护，以及如何安排培训的初步概念。这些概念文档应该允许在产品架构开发的过程中发生演变，这样就可以在架构定义期间反映出制定的产品架构决策。在软件实现期间，应安排一个平行的工作来实现和测试这些开发后的过程，以便当产品成功通过验收测试时它们就可使用。

8）软件需求追踪矩阵。需求追踪矩阵应初步确定每个需求的来源和它对计算环境需求的依赖。软件需求的来源可以包括，例如，涉众、法律规章、标准惯例、公司政策或指导方

⊖ 分布式计算环境，软件技术路标，http://www.sei.cmu.edu/str/descriptions/dce.html

针、业务模型或分析驱动。该矩阵可以在整个软件开发后的过程（销售、培训和维护）中演化。在这个需求定义阶段期间，为软件产品和接口指定的需求应可以追溯到它们的来源，例如，客户需求陈述、工作陈述、项目许可文档、市场调研、权衡分析结果、建议和决策。该矩阵可以作为一个单一的产品来维护，也可以为确定的每个软件配置项将它分割为两个或更
294 多的矩阵来维护。

17.2　软件工程集成产品团队（软件需求定义阶段）

SWE-IPT 应由高级软件工程师建立和领导。SWE-IPT 是由软件专家和主题相关专家组成的技术工作团队，他们都将对软件工程任务有所贡献。SWE-IPT 应该是一个多学科的团队，它的成员代表了各个软件学科，根据具体情况，包括软件工程、软件实现、计算环境实现、软件开发后的过程实现、软件测试和评估、软件开发管理、安全性以及人员与系统集成。

1）*开发业务模型*。SWE-IPT 应该安排软件需求分析任务以捕获涉众的需求和约束，因为它们与开发软件的业务过程或嵌入式系统过程相关。涉众[⊖]主要包括客户、市场人员、业务管理人员、合伙人、供应商和转包商。应该开发业务模型来描述贯穿业务过程的各种线程，包括用于管理如何实施过程以及过程如何应对可能遇到的各种情况的业务规则。

2）*开发初始功能行为模型*。SWE-IPT 应该进行功能分析和分配，以对给软件产品的抽象或具有挑战的操作任务进行分解。功能分析为软件产品和计算环境的要素如何合作来实现操作过程制定了一个更详细的模型或表示方法。通过将高层次功能分解为低层次功能，按逻辑顺序排布功能，以及从高层次到低层次为功能分配性能需求，功能分析可以完成模型的制定。功能间的数据流应该从业务模型中确定的数据流中获取。抽象的操作数据应分解为软件数据元素的集合，并且赋予每个元素唯一的标识。

3）*综合概念设计方案*。SWE-IPT 应该安排软件设计综合，从而为物理架构的结构确定初始概念。*初始的结构概念应该确定初始的结构组件和它们的接口。考虑到一些相似或通用功能需求分配至一个结构组件，结构概念的每个元素都应该可追踪到其功能架构。结构化概念可以用产品层次或产品框图来表示。产品框图应该描述产品的结构分布、结构组件间的
295 内在接口以及与计算环境要素交互的外在接口。

如果有必要，初始的结构概念可以制定多个软件配置项以提供一个令人满意的解决方案。其他可选的结构概念也应该加以考虑，且可行的结构应在系统分析期间通过权衡分析和风险评估进行评价。若确定了多个软件配置项，为了反映架构决策，必须对项目计划、工作分解结构和规约树进行修改。随着物理结构逐渐成熟，功能架构或业务模型的有效性或正确性可能面临挑战。此外，产品物理结构还可能影响计算环境需求的设计。这些冲突必须在问题报告、确定的可行方案以及针对项目目标进行评估的方案、产品质量度量和风险中加以捕获。

***注**：此处，本书描述的过程忽略了计算环境的设计而致力于软件产品的设计。为了在计算环境以及软件产品需求、性能之间达到一个合适的平衡点，需要考虑计算环境和软件产品之间的权衡。因此计算环境实现团队开展一个类似的方法来综合和分析计算环境设计的可选方案就很有意义了。既然这本书是关于软件工程的，计算环境的设计和实现没有详细描述。

⊖ 涉众还包括软件开发组织的每个成员以及项目管理团队的代表人员。

4）分析产品的可选方案、冲突和权衡。SWE-IPT 必须评估已确定的软件需求和功能、物理架构之间的冲突，以确定一个首选的架构解决方案。架构候选方案必须经过分析，才能确定它们是否可以在项目成本和进度计划之内实现，以及确定它们与涉众需求和期望之间的冲突。应该对架构候选方案进行优先排序并将首选的解决方案推荐给 SWE-IPT。应该对首选的设计方案进行分析，确定它代表的是软件需求、功能以及物理架构的一致结合。

对于首选的设计方法，为了了解它在实现上面临的挑战并确定采用该方案的所有内在风险，应对其进行分析和评估。必须对已确定的风险进行评估，设计消除、避免或将风险降低到可接受级别的方法。在采取一个设计方案之前，必须对风险进行确定、衡量和缓解。采取有内在风险的架构决策会将软件开发项目置于危险之中。项目后期的架构修改将导致更大的代价和潜在可能的进度延迟。为了获取风险鉴定的结果，包括风险发生的概率和风险给项目带来的后果，必须准备风险评估报告。如果首选的架构方案不影响项目和技术方案，则可以 296
采用该方案作为架构设计决策。会给技术计划范围带来不利影响的架构方案必须在软件变更提议中进行文档化。为了得到授权允许，变更提议必须提交给技术 CCB。如果一个变更提议对项目计划的影响超出了技术 CCB 的职权，则必须将它提交给项目 CCB 以得到授权。这些软件变更提议代表的是项目级别的调整，需要额外的资源以及项目方案、进度安排和资源分配的修改。

5）制定软件需求分配。随着业务模型逐渐成熟，应加强功能和物理架构以便产品需求在项目目标内可以实现。为了在软件配置项、计算环境和软件产品接口上分配需求，应该利用业务模型。SWE-IPT 应该为这些软件架构元素准备需求规约。SWE-IPT 还应准备需求追踪矩阵以将涉众需求和期望、项目目标和操作过程的各方面与架构元素间分配的需求关联起来。

6）准备软件开发后的过程概念。SWE-IPT 应该为软件开发后期的每个过程评估需求。SWE-IPT 还应该应用系统工程实践来为软件开发后期的每个过程制定初始操作概念。每个软件开发后的过程的概念文档应该阐述过程范围、其操作行为以及初始的功能和物理架构。

7）准备并撰写风险缓解计划。SWE-IPT 必须为每个已确定的风险准备风险缓解计划。必须对风险进行持续监控，直到风险被消除。风险缓解计划应该确定风险监控的方法、应急计划激活的标准以及风险确实不可避免时执行的一连串操作。

8）修订工作分解结构。SWE-IPT 必须评审并更新工作分解结构，以反映架构决策和被采用的变更提议带来的影响。应该对工作包、相关任务和资源分配进行调整来反映对设计、实现和测试软件产品及开发后的过程所需的工作的更深层理解。一些任务可能缩小其范围，另一些任务可能需要更多的时间和资源，还可能需要创建一些新的任务。WBS 应该是一个灵活的机制，它可以从初始规划预估开始调整，以反映架构决策和被采用的变更提议。

9）精化产品规约树。由于确定了软件配置项，必须对软件资料的需求进行重新审视，以便规约树与配置项采用的架构结构一致。对于每个确定的配置项，规约树应该反映其所需资料的软件层次。产品规约树必须进行扩展和更新以反映管理每个配置项开发所需的软件计 297
划、规约、文档、模型、图表或其他形式的文档。

10）精化项目和技术计划。应该对软件开发项目其他阶段的计划进行重新审视，以反映对开发工作范围的更深理解。项目和技术计划必须是动态可变的文档，可以对其持续更新来反映架构决策和授权的变更提议。项目计划必须阐明剩余待执行的工作范围，并详述它们成功执行的概率。

11）准备软件术语注册表。SWE-IPT 应该准备软件术语注册表[⊖]，为软件架构元素指定唯一的识别、名称和定义。注册表条目应该包括业务模型确定的元素，以及初始的功能和物理架构。数据项应该根据它们的目标、类型（如 constant，variable，string，integer，Boolean 或 date）、安全性分类（如适用）、度量单位和值的可接受范围进行定义。术语注册表是为了确保整个软件开发团队知晓已授权的架构元素的名称、识别和定义。这是为了确保架构元素能够被恰当地使用，并确保不会赋予架构元素重复的名称或标识。

12）准备软件需求评审（*software requirement review, SRR*）。SRR 是一个正式的项目级别的评审，开展 SRR 可以向项目管理专家、客户和其他涉众展示软件架构的状态。SWE-IPT 应该为 SRR 作好准备。已经识别的风险和它们的缓解计划必须经过评审。最后，应该推进对 WBS、规约树和项目计划的修改。应该确定驱动项目计划发生主要修改的架构决策并追踪它对 WBS、规约树和项目计划的影响。软件更新的技术计划以及软件质量检验和审核的结果应该进行报告。

17.3　软件实现（软件需求定义阶段）

1）参与 SWE-IPT。软件实现团队的高级代表应该参与 SWE-IPT，帮忙制定建设性的软
298 件需求规约和分配决策。软件实现团队的代表将他们有关计算语言、设计模型以及实现挑战的知识技能贡献到软件工程任务中。

2）确定软件实现的挑战、约束、可行性和风险。软件实现团队的代表应该确定实现面临的挑战、约束以及与已制定的软件需求相关的风险。为了确保为软件产品制定的需求能够实现，这些代表是必不可少的。

3）确定软件开发环境。软件实现团队应该确定软件开发环境的要素，这些要素对演化的软件架构的实现、测试和调试是非常必要的。

17.4　计算环境准备（软件需求定义阶段）

1）参与 SWE-IPT。计算环境团队的高级代表应该参与 SWE-IPT，帮助制定建设性的软件需求规约决策。计算环境团队的代表将他们有关计算硬件、网络、交互、操作系统、中间件和计算技术挑战的知识技能带入软件工程任务中。

2）确定计算环境实现的挑战、约束、可行性和风险。计算环境团队的代表应该确定计算技术的难点、约束以及与已制定的计算环境需求相关的风险。为了确保为计算环境制定的
299 需求能够实现，计算环境实现团队的代表是必不可少的。

17.5　开发后的过程实现（软件需求定义阶段）

1）参与 SWE-IPT。软件开发后的过程实现团队的高级代表应该参与 SWE-IPT，帮助制定建设性的软件需求规约决策。软件开发后的过程领域的代表将他们有关软件复制、销售、培训和维护技术、过程和挑战的知识技能带入 SWE-IPT 中。

2）确定开发后的过程实现的挑战、约束、可行性和风险。软件开发后的过程实现团队的代表应该确定实现面临的挑战、约束和与开发后的过程概念相关的风险。为了确保为软件产品制定的需求和维护环境不会约束开发后的过程，软件开发后的过程实现团队的代表是必

⊖ 原本指的是一个数据词典。然而，有必要在命名注册表中解决全方位的架构元素。

不可少的。

3）准备软件开发后的过程的概念。软件开发后的过程实现团队的代表应该开发操作概念（concept of operation，CONOP）文档，操作概念文档必须准备好以便为每个过程领域确定初始范围和需求。CONOP 应该确定支撑这些过程必须的工具、器材、设备或者材料。

17.6　软件测试和评估（软件需求定义阶段）

1）参与 SWE-IPT。软件测试和评估团队的高级代表应该参与 SWE-IPT，帮助进行建设性的软件需求规约决策。这些代表将他们有关压力计算负载的重现、资源利用率的度量以及其他与软件测试相关的挑战的知识技能带入 SWE-IPT 中。

2）确定软件测试和评估的可行性、挑战、约束和风险。软件测试和评估团队的高级代表应该确定实现面临的挑战、约束以及与软件测试和评估工作相关的风险。为了确保所制定的软件产品需求在分配的资源和约束下是可测试的，软件测试和评估团队的代表是必不可少的。

3）准备软件测试计划。软件测试和评估团队应该准备测试计划，描述安排软件验收测试的策略。该计划应该确定初始的测试策略和测试用例、软件测试环境的要素，并基于软件规约制定测试进度安排。

4）修订软件质量保证计划。软件测试和评估团队应该准备软件质量保证计划，描述软件概要架构定义期间安排的软件检验和审核的策略。

5）进行软件质量保证检查和审核。应该定期地安排软件质量检验（在软件需求定义阶段期间），评估软件需求规约和架构产品。应安排下列这些检验：

- 业务模型的检验。
- 功能架构的检验。
- 物理架构的检验。
- 产品需求规约的检验。
- 计算环境规约的检验。
- 软件接口规约的检验。
- 软件测试计划的检验。
- 需求追踪矩阵的检验。
- 开发后的过程概念文档的检验。
- 术语表的检验。

300

为了确保团队能够遵从制定的政策和过程，应该安排软件质量检验。还应在 SRR 之前安排软件审核，以确保可用的软件规约能为软件需求的评估提供一个一致且可追踪的框架，其中软件需求的评估是依据项目和技术方案，以及涉众需求和期望开展的。应安排下列这些审核：

1）软件需求审核——从业务模型到概要的功能和物理架构，追溯涉众需求至软件规约中制定的需求。导出需求的来源应该与工程分析或识别隐含需求的权衡分析关联起来。导出的需求必须包含在需求追踪矩阵中。

2）软件测试审核——追溯软件测试用例或场景至业务模型的数据处理事务和每个场景将执行的软件需求。测试用例应该在需求追踪矩阵中被恰当地反映出来。

3）校正行为的审核——追溯每个软件问题报告、变更请求或提议至它的权衡分析报告

和更正措施计划，确保校正行为在业务模型、软件需求规约、功能和物理架构以及受措施影响的相关架构产品中被恰当地反映出来。

17.7　评审、里程碑和基线（软件需求定义阶段）

1）安排 SRR。为了向涉众证明软件规约和开发后的过程概念的充分性，应安排软件需求评审。SRR 应将重点放在需求分配到软件产品、它的配置项、外在接口以及计算环境要素上的状态。应该重点评审软件需求定义期间制定的架构决策和已经确定的所有风险。评审的目的不是让未参与软件工程活动的人员开讨论会来批准或理解软件需求；而是为了确定软件需求是完整而且与涉众需求和项目目标相一致的。为了达到这个目的，有必要证明已经充分研究过操作问题和解空间，分析过可选的解决方案，以及有可靠的技术原理制定架构决策。

301 典型的软件需求评审议程应该解决以下问题：

1. 软件需求定义阶段——目标
   1.1. 需求分析产品
      1.1.1. 涉众需求和约束
      1.1.2. 业务模型组合
      1.1.3. 概要功能架构构造（行为模型和层次）
      1.1.4. 概念物理架构
      1.1.5. 关键权衡分析、可选方案、结果、决策和基本原理
      1.1.6. 软件需求规约状态
      1.1.7. 计算环境规约状态
   1.2. 开发后的过程概念
      1.2.1. 复制过程
      1.2.2. 销售过程
      1.2.3. 培训过程
      1.2.4. 支持过程
      1.2.5. 开发后的过程开发
         1.2.5.1. 进度安排和里程碑
         1.2.5.2. 资源需求
   1.3. 软件工程计划、进度安排和里程碑（概要架构定义阶段）
2. 软件实现策略
   2.1. 软件实现的挑战、约束和风险
   2.2. 软件开发环境
   2.3. 软件实现的进度安排和里程碑（概览）
3. 计算环境定义
   3.1. 计算环境实现的挑战、约束和风险
   3.2. 计算环境实现的进度安排和里程碑（概览）
4. 软件测试和评估
   4.1. 软件测试和评估的挑战、约束和风险
   4.2. 软件测试计划（概览）

4.2.1. 软件测试策略
4.2.2. 软件测试环境
4.2.3. 软件测试用例
4.2.4. 测试和评估的进度安排和里程碑
4.3. 软件质量保证计划
4.3.1. 软件质量检验
4.3.2. 软件质量审核
4.3.3. 软件质量的进度安排和里程碑
5. 软件控制
5.1. 更新的项目计划和进度安排
5.2. 更新的规约树
5.3. 更新的工作分解结构
5.4. 工程工件库
5.5. 变更历史库
5.6. 技术风险库
5.7. 软件术语表

2）制定需求基线。当软件需求评审成功完成时，SWE-IPT 应该将软件需求规约置于技术配置控制下。这些规约组成了架构需求基线，并确立了产品功能和物理架构的开发基础。功能基线是由软件产品、接口和计算环境规约组成的。一旦需求规约确立了，对这些规约的变更需要技术 CCB 对变更请求或提议的批准才能实施。

既然业务模型是导出软件需求的基础，它应该在软件工程工件库中加以捕获。概要的功能和物理架构不应在此时加以控制。功能和物理架构在概要架构定义阶段将继续演变，并为软件架构解决方案的扩展提供技术基础。

302
~
304

第 18 章

Software Engineering: Architecture-Driven Software Development

软件架构定义

软件开发过程中软件架构定义阶段的目标是建立软件产品的功能和物理架构，并启动软件开发后的过程的开发。架构定义阶段被分为两个子阶段，如图 18-1 所示。第一阶段，概要架构定义，主要将顶层功能分解为一个完整的功能层次。它还导出对结构配置的概念性组
305 件进行定义的初步物理架构。第二阶段，详细架构定义，通过建立结构单元和组件规约以及软件集成策略来最终确定物理架构。在软件架构定义阶段，应该建立计算环境的定义来设计一个统一的软件产品架构。

图 18-1　架构定义阶段

在概要架构定义阶段，对顶层功能进行分析并将其分解为一个组件和单元构成的完整功能层次。没有固定的分解层次的数目。然而，分解应该继续进行直到功能解决方案完整简洁易于理解。当一个功能单元被认为是一个简单的数据处理动作时，就不需要进一步的分解了。简单的元素应以功能单元标记，表示架构的基础部分。功能性单元规约应该能够反映行为模型所确立的功能特性。物理架构定义从构思概念结构解决方案开始。该结构解决方案表明概念组件的组织结构，这些组件整合了最终由一系列底层结构元素集合提供的主要数据处理功能。该开发阶段以概要设计评审（preliminary design review，PDR）结束。PDR 旨在表明功能分解和行为模型是完整并遵循软件需求的。

306 在详细架构定义阶段，必须通过围绕着一致主题调整和组织功能单元来构思最底层结构

单元。结构单元必须就其行为、数据元素、算法、条件控制机制、接口和故障检测与恢复过程等进行说明。这个工作可通过合并功能单元规约并解决冲突或冗余的功能特性来完成。应该构造结构单元继承层次以规划如何将面向对象特性从父亲单元向子代传播。必须建立软件集成策略来弥合结构单元和概念组件间的设计鸿沟。必须建立结构化用户接口层次以确定信息展示、导航控制、数据格式化和数据输入的机制。必须对数据库查询进行说明，它满足确定的功能数据库事务行为。软件测试流程和软件实现计划必须被最终确定。

详细架构定义阶段以关键设计评审（critical design review，CDR）结束。CDR 旨在表明物理架构是完整、已验证和确认的。物理架构必须依据功能架构和软件需求基线进行验证。所提取的结构单元规约将会指导软件实现团队对各个软件单元进行设计、编码和测试。结构单元和组件的分配必须进行评审来以确保结构配置具有合理的复杂度并将有助于问题的解决和预期的产品改进（P^3I）。

18.1　概要架构定义

概要架构定义着重于建立功能架构和概念化结构解决方案。该阶段的目标是充分理解诸如涉众需求、软件规约和计算机技术能力等所包含的软件挑战。功能架构提供对软件产品运行要素的分析解决方案。软件产品结构的初始配置将被建立为主要的顶层结构组件的概念化表示。初始结构配置包括抽象组件和用户接口设计机制的分配及其之间的相互连接。

18.1.1　概要架构定义的产品

在软件开发项目的概要架构定义阶段应该生成以下产品：

1）*功能行为模型*。功能行为模型提供了软件解决方案的功能流、控制流、数据流、时序、错误检测与处理流程和资源利用等方面的特性。行为模型是从业务模型导出的并将操作活动分解为软件产品将被设计具有的行为。

307

2）*功能层次*。功能层次确定了将功能复杂性分解为基本简单功能的层次。顶层的功能组件被分解成较低层次的组件以提供用来导出软件行为的逻辑框架。较低层次的功能组件必须进一步分解直到可识别的功能单元。功能单元表示不能再分解或者效率分析不支持进一步探究或分解的基本功能。

3）*数据库事务行为图*。该图提供软件产品必须支持的每个数据库事务的行为表述。这些图应该确定所访问的数据库表、事务查询的预期响应和事务回滚的过程。

4）*用户界面功能层次*。该层次描述了用户界面功能机制如何被组织为功能组件、子组件和最终的功能单元。

5）*概念组件框图*。该图提供了概念组件、组件间的接口以及与操作者和计算环境元素间接口的布局。

6）*软件测试用例*。软件测试用例应该通过追踪业务模型中行为的操作线程来确定。每个操作线程代表了一个唯一的测试用例。每个测试用例应该识别执行数据处理事务所需的条件和判定事务处理是否满足的准则。

7）*更新的需求追踪矩阵*。必须更新需求追踪矩阵以反映功能架构、物理架构和软件测试用例的演化元素。

8）*更新的术语表*。必须更新软件架构字典以反映功能和物理架构的元素。

9）*初步软件实现计划*。应该草拟软件实现计划以建立完成软件开发阶段所需要的工作

量和资源投入量的基准。实现计划应提供一个细化相关工作包以反映演化的软件架构并建立子包间资源分配的基础。此外，软件实现计划应该确定在软件实现团队中人员的需求，包括正确实现软件架构所需的技能类型。

10）初步计算环境实现计划。应该草拟计算环境实现计划以建立完成计算环境所需工作量和资源投入量的基准。计算环境实现计划将会提供一个细化相关工作包以反映计算环境实
308 现团体的资源需求的基础。

11）修订的软件测试计划。软件测试和评估团队必须修订软件测试计划以反映准备和进行软件验收测试的相关工作。该软件测试计划应该提供一个细化工作包以确定软件测试和评估团队的资源需求的基础。

12）初步的开发后的过程实现计划。开发后期实现团队必须准备初步的开发后的过程实现计划以反映与实现开发后的过程相关的工作。开发后的过程实现计划将提供一个细化相关工作包以确定开发后的过程实现团队资源需求的基础。

18.1.2　软件工程集成产品团队（概要架构定义阶段）

1）细化业务模型。软件工程集成产品团队应该分析每一个从业务模型中导出的顶层功能组件以决定如何分解功能复杂性。应该对复杂功能组件的行为进行分析以深入了解为支持该操作过程所必须提供的软件功能行为。应该准备好功能行为模型以辅助对复杂功能进行分析。应该确定能让功能组件正确运行并检测和响应失效条件所需的导出功能。

2）细化功能层次。SWE-IPT应该进行功能分析以分解功能组件从而建立一个更详细的功能层次。功能组件应该被分解为功能子组件，直到功能单元能够被识别。功能分解的可选方案应该进行评估和权衡研究以确定与每个可选方案相关的性能、接口和风险。首选的功能解决方案应该被集成到功能层次并反映到功能架构的其他工件。

3）综合概念化配置可选方案。SWE-IPT应该进行软件设计综合以识别顶层概念组件。应该识别并评估概念化配置可选方案以将选择范围缩小至一个或多个更实用的可选方案。

4）分析功能候选方案、冲突和权衡。SWE-IPT应该评估已识别的功能分解、分组和分配的可选方案以确定最佳解决方案集。应该识别并评估与每个可选方案相关的风险以辨别出能排除、避免或降低风险达到可接受程度的架构设计方案。应该选择的首选解决方案应该提
309 供最好的性能，能满足涉众的需求和期望，有机会实现项目成本和进度的目标。SWE-IPT应该评估已识别的功能和初始物理架构间的冲突以确定潜在的纠正措施。解决架构冲突的可行方法应该依据程序目标和涉众需求来进行研究和评估。必须识别与每个方法相关的软件实现和测试的挑战和风险。应该对可选方法进行优先级排序以为架构决策提供便利。

5）说明功能架构。SWE-IPT应该进行软件需求分析以一致地说明功能组件和单元的需求。为了发出和接收功能元素必须确定内部软件接口或数据交换需求。必须指定控制、错误处理和资源管理机制来排除无效的方案。

6）验证功能架构。随着功能架构逐渐成熟，必须对其进行验证以确保其反映满足需求基线的解决方案并在项目目标内可实现。当调整需求基线和功能架构时，必须对需求追踪矩阵进行更新以反映软件需求如何分配到功能架构的元素中。需求追踪矩阵应将功能架构元素与需求基线和测试用例元素相关联。

7）更新风险缓解计划。应该为那些不能够消除或者避免，仍然威胁着项目目标完成的风险准备风险缓解计划。风险评估报告应该记录每个风险评估的结果，包括发生的概率和风

险一旦发生的后果。风险缓解计划应该确定监控和阻止风险发生所采取的行动方案，造成风险按计划进行不可接受的准则和一旦符合风险偏离准则时应采取的应急行动。

8）修订技术方案。需要对工作包中确定的任务进行重新检查和细化以准确反映接下来要执行的工作。应该对技术方案进行修订，必须对程序工作分解结构、工作包和资源分配进行调整以反映对开发工作剩余部分的进一步理解。

9）细化项目计划。应该对软件开发项目的剩余阶段的项目计划进行更新以反映开发工作的剩余范围。项目计划必须是活的文档并反映对剩余工作范围已作的设计决策。

10）更新软件术语注册表。应对软件术语注册表进行更新和扩展以反映功能架构的元素。

310

11）为 PDR 作准备。SWE-IPT 应为 PDR 作准备。PDR 的目标是向项目管理方、客户和其他涉众提供一致的软件需求、功能架构和概念化结构配置。要确定引导功能架构形成的架构决策。必须追踪架构决策对 WBS、项目和技术方案的影响。应对在概要架构定义阶段实施的软件质量保证检查和审核进行报告。要对在此开发阶段识别的风险及其风险降低计划进行讨论。

18.1.3　软件实现（概要架构定义阶段）

1）参与 SWE-IPT。软件实现团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的架构决策。来自软件实现的代表将他们关于实现语言、设计模式和软件实现挑战的知识带给 SWE-IPT。

2）识别软件实现的挑战、约束、可行性和风险。软件实现团队的代表应该识别与理解功能结构相关的实现挑战、约束、可行性和风险及其对软件实现工作的影响。

3）准备初步软件实现计划。软件实现团队应准备初步软件实现计划以反映在概要架构定义阶段获取的信息。这个计划必须确定完成软件实现工作的软件实现任务、工作包和进度里程碑。由于结构配置的不确定状态，这个计划可能不会具有执行所需的必要的明确性。然而，它应该提供比之前的计划版本更准确的对预期工作量的预测。

18.1.4　计算环境准备（概要架构定义阶段）

1）参与 SWE-IPT。计算环境团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的架构决策。来自计算环境团队的代表将他们关于计算机硬件、网络、通信、操作系统、中间件和其他运行挑战的知识带给 SWE-IPT。

2）识别计算环境实现的挑战、约束和风险。计算环境团队的代表应该识别与整合计算环境实现工作量与需求基线和功能架构相关的计算挑战、约束和风险。

311

3）准备初步计算环境实现计划。计算环境团队的代表应准备计算环境实现计划。这个计划必须识别用于建立计算环境的实现任务、工作包和进度里程碑。由于结构配置的不确定状态，此计划可能不会具有执行所需的必要的明确性。但是，它应该提供比之前计划版本更准确的对预期工作量的预测。

18.1.5　开发后的过程准备（概要架构定义阶段）

1）参与 SWE-IPT。开发后的流程团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的架构决策。来自开发后的过程团队的代表将他们关于软件复制、销售、训练、客户和

软件支持挑战的知识带给 SWE-IPT。

2）识别开发后的过程实现的挑战、约束和风险。开发后的过程团队的代表应该识别与整合开发后的过程与需求基线和功能架构相关的实现挑战、约束和风险。

3）准备初步开发后的过程实现计划。开发后的过程团队代表应准备开发后的过程实现计划。此计划需确定用来建立销售、培训和软件维护过程的开发后的过程任务、工作包和进度里程碑。由于结构配置的不确定状态，此计划可能不会具有执行所需的必要的明确性。但是，它应该提供比之前计划版本更准确的对预期工作量的预测。

18.1.6　软件测试和评估（概要架构定义阶段）

312

1）参与 SWE-IPT。软件测试和评估团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的架构决策。来自软件测试和评估的代表将他们关于测试需求和挑战的知识带给 SWE-IPT。

2）识别软件测试和评估的挑战、约束和风险。因为组件架构是通过软件工程过程导出的，软件测试和评估团队的高级代表应该识别与需求基线和功能及物理架构可选方案相关的实现挑战、约束、可行性和风险。

3）准备软件测试计划。软件测试和评估团队必须细化软件测试计划。应通过识别业务模型中的测试线程导出软件测试用例。一个测试用例描述了初始的测试环境状态；在测试过程中的输入、动作或事件；预期的软件响应或数据处理事务的结果。该计划必须确定用以建立软件验证测试环境和过程、限定测试环境和执行验收测试的测试和评估任务、工作包和进度里程碑。由于结构配置的不确定状态，该计划可能不会具有执行所需的必要的明确性。但是，它应该提供比之前计划版本更准确的对预期工作量的预测。

4）进行软件质量保证检查和审核。应该定期进行软件质量检查以查验软件工程工件和产品从而确保其完整性、准确性以及与已有策略和过程的一致性。应该进行软件质量评审以确保团队遵循已建立的流程和已认可的计划。在初步软件架构定义工作中应该实施以下软件质量检查：

- 检查功能层次。
- 检查功能组件规约。
- 检查功能单元规约。
- 检查概念配置文档。
- 检查团队的技术方案。
- 检查软件术语表。
- 检查需求追踪矩阵。

软件审核应该先于 PDR 以确保软件产品和架构工件是完整的并整合了已认可的变更提议和请求。应进行以下审核：

1）功能架构审核。追踪从最初来源（涉众需求）到业务模型、需求基线和功能架构的功能元素需求。导出功能元素的来源应可追踪到架构决策或工程分析结果。

2）测试用例验收审核。追踪每个测试用例到其要确认的规约需求。一个测试用例必须能从一个操作线程（业务模型），到需求来源（涉众的需求），到其想要确认的规约需求（规
313 约标识符）来进行追踪。一个软件测试可能影响一个或多个架构规约（需求、接口或功能），这应该恰当地反映到需求追踪矩阵中。

3）校正行为审核。追踪每个已认可的变更请求或提议到其纠正措施的配置。该审核必须确保纠正措施被恰当地完成并反映在受影响的软件产品和架构工件中。

18.1.7　评审与里程碑（概要架构定义阶段）

应该以描述功能架构和架构决策的落实为目标来实施 PDR。PDR 旨在将软件架构落实为一个使得软件架构能够随时间演化的框架。评审的重点应该是设计决策的理论基础以及它们如何影响项目计划和项目目标的实现。架构概要设计评审的议程应包含如下几个主题：

1. 概要设计评审——概述
   - 1.1. 需求基线和变更提议状态
   - 1.2. 功能架构状态
   - 1.3. 物理架构状态（概念结构配置）
   - 1.4. 关键权衡分析、可选方案、结果、决策和原理
   - 1.5. 需求追踪矩阵
2. 软件实现
   - 2.1. 初步软件实现计划
   - 2.2. 初步软件实现进度表和里程碑
   - 2.3. 软件实现的挑战、约束、可行性和风险
3. 计算环境
   - 3.1. 计算环境实现计划
   - 3.2. 计算环境评定计划
   - 3.3. 计算环境进程表和里程碑
   - 3.4. 计算环境的挑战、约束、可行性和风险
4. 软件测试和评估
   - 4.1. 软件测试和评估计划
   - 4.2. 软件测试进度表和里程碑
   - 4.3. 软件质量保证检查
   - 4.4. 软件质量保证审核
   - 4.5. 软件测试和评估的挑战、约束、可行性和风险
5. 软件开发后的过程
   - 5.1. 软件开发后的过程计划
   - 5.2. 软件开发后的过程进度表和里程碑
   - 5.3. 软件开发后的过程的挑战、约束、可行性和风险

314

18.2　详细架构定义

详细架构定义阶段关注软件架构的完成和向软件实现的转换。通过识别集成结构组件的方式可建立顶层概念级和导出物理单元的联系，从而建立物理架构。物理架构应该以某种允许修改、扩展和加强以减少软件支持成本和有助软件重用的方式来进行配置。

18.2.1　详细架构定义的产品

物理架构应该锚定在结构单元的构造上。通过合成同化的功能单元规约和解决冲突和冗余需求来建立结构单元规约。软件集成策略必须通过合成一个或多个层次的结构组件来导出，这些组件又集成了多个结构元素来匹配顶层概念结构。详细架构定义阶段产生如下产品：

1）结构单元框图。构造结构单元框图来表示结构单元如何相互作用。框图是一个以较简单的方式来解释复杂系统的方法。它们由表示结构单元的标记框组成，框间通过指示从输入到输出框的数据流方向的箭头连接。

2）结构单元继承层次。应该通过将类似结构单元分组并派生出父单元的特征来开发结构单元继承层次。它应该展示出子结构单元如何继承一般功能与数据元素以及每个子元素如何添加独特的特征。这在面向对象领域称为特化。

3）软件集成层次。应该构造软件集成层次以描述结构单元如何被装配并集成为较大的组件。应该对结构组件进行定义，它包含一个集成的组件和支持集成测试所需的测试桩。集成层次应该描述得到完整的集成软件配置项所需的集成顺序层次。该层次也应该描述在层次结构的哪部分进行集成测试。

4）物理用户接口层次。应该通过分组相关用户接口机制，将其合成为结构单元和组件，并配置物理用户接口来构造物理用户接口层次。

5）数据库结构框图。数据库结构框图提供数据库表、记录、域和连接多个表间记录的关系的图形表示。

315

6）数据库查询规约。应该开发数据库查询规约以为通用的数据库事务提供通用的查询指令。在功能分析阶段确认的数据持久性功能应该是数据库查询规约的基础。每个数据持久性功能涉及数据在库中的存储或数据库管理系统（dataBase management system，DBMS）。结构化查询语言（structured query language，SQL）指令提供一个在 DBMS 中对信息进行增加、删除、更新和排序的指令或命令的标准集。数据库查询规约应该建立数据库特权查询集，它提供对受限数据库信息的访问。

7）结构单元规约。应该使用特定的语言表达式或结构来详细说明每个结构单元以便对其进行设计。应为每个结构单元建立软件开发文件夹（SDF）从而为所有后续实现工件提供知识库。与每个物理单元相关的需求规约应保存在适当的 SDF 中。结构单元规约应说明每个单元必须遵循的联合功能需求、接口和程序设计特征（例如，代码行预算）。

8）更新的需求追踪矩阵。应该对需求追踪矩阵进行更新以反映结构单元与功能架构的关联以及软件集成策略的作用范围。

9）更新的软件术语注册表。应该对软件术语注册表进行更新以反映物理架构并确保软件架构中每个结构单元、组件和数据元素的名称和定义是唯一的。

10）敲定软件实现计划。应该敲定软件实现计划以确定完成软件开发工作实现阶段所需的工作量和资源量的基准。实现计划必须细化工作包以反映结构配置和软件集成策略的演化。

11）敲定计算环境实现计划。应该敲定计算环境实现计划以确定建立和验证在软件验收测试中所使用的计算环境所需的工作量和资源量的基准。计算环境实现计划必须细化工作包以反映计算环境实现团队的资源需求。

12）软件测试计划。软件测试和评估团队必须扩展软件测试计划以建立测试场景和过程。软件测试计划必须细化工作包以反映与准备和执行软件验收测试相关的工作。

13）开发后的过程实现计划。开发后的过程实现团队必须扩展开发后的过程实现计划以反映与实现开发后的过程相关的工作。开发后的过程实现计划必须细化工作包以反映与支持 316
每个后期工作过程的环境的定义、设计、装配、集成和确认相关的工作量。

14）软件技术数据包。必须准备技术数据包从而为软件实现和验收测试提供基础。软件 TDP 至少应包括以下架构工件：

- 软件需求基线（软件产品、接口规约和计算环境规约）。
- 结构单元规约。
- 结构组件规约。
- 结构装配规约。
- 软件集成策略。

18.2.2　软件工程集成产品团队（详细架构定义阶段）

1）综合结构单元候选方案。SWE-IPT 应进行软件设计综合以识别和分组通用功能并分配至已定义的结构单元。这包括决定物理单元继承层次和物理用户接口层次。应该构造结构单元框图以确定结构单元间应该如何连接。应该识别和评估候选的结构单元配置以将解空间缩小至最优的物理设计方案。

2）开发软件集成层次。应该开发软件集成层次以描述软件集成策略。该层次描述了结构元素将会如何进行装配、集成和测试以形成一个独立的集成结构配置项。

3）分析结构配置候选方案、冲突和权衡研究。SWE-IPT 应该分析结构设计可选方案以确定首选的架构解决方案。必要时，应该对架构设计候选方案进行权衡研究。必须识别和评估架构设计可选方案的风险以确保架构配置结果可以在项目成本和进度约束内实现。应该选择能够平衡性能特性、减少物理架构复杂度、稳定架构和满足涉众的需求和期望的最优结构设计解决方案。SWE-IPT 应该评估所识别的功能和物理架构间的冲突以确定潜在的纠正活动过程。应该依据项目目标和涉众需求来研究和评估解决架构冲突的可行方法。必须确认与每个方法相关的软件实现和测试的挑战和风险。应该对候选方法进行优先排序从而为架构决策的制定提供便利。 317

4）更新风险缓解计划。应该为那些不能被消除或避免并仍然威胁着项目目标实现的物理单元风险准备风险评估记录。应该为每个已识别的风险制定风险缓解计划。风险评估记录应该记录风险评估的结果，包括发生概率和风险一旦发生的后果。风险缓解计划应该定义监控和阻止风险发生的行为，判断风险已严重到不能继续按计划进行的准则和应采取的应急计划。

5）修订 WBS。一旦物理架构完成而且软件实现计划敲定，应该对 WBS 进行调整以反映对软件实现、测试和评估的工作范围的进一步了解。与软件工程相关的工作包、任务和资源分配必须符合软件实现计划中定义的任务。

6）细化技术方案。必须为软件开发程序中的剩余阶段再次查看技术方案以使其反映剩余的工作范围。技术方案必须是活的文档并反映制定的设计决策和它们对要执行的工作范围的影响。与软件工程相关的工作包、任务描述和资源分配必须符合计划中确定的任务。软件工程计划结果应该确定在软件开发工作剩余阶段中应该执行的任务。SWE-IPT 必须综合团

队的计划并细化集成技术方案和进度表。

7）更新软件术语注册表。应该更新和扩展软件架构字典以反映物理架构。所有物理单元和组件的名称必须添加到软件架构字典中。

8）准备 CDR。SWE-IPT 必须为架构的 CDR 作准备。CDR 的目的是向项目管理方、客户或其他涉众展示一致的需求的基线和功能与物理架构。应该对驱动项目结构和计划重大变更的设计决策进行识别并追踪其对 WBS 和项目计划的影响。还应该考虑在详细架构定义期间执行的软件测试计划以及质量保证检查和审核。应该对已识别的风险及其风险缓解计划进行评审。

18.2.3　软件实现（详细架构定义阶段）

1）参与 SWE-IPT。软件实现团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的功能和物理架构决策。来自软件实现团队的代表将他们关于实现语言、设计模式和软件实
318 现挑战的知识带给 SWE-IPT。

2）识别软件实现的挑战、约束、可行性和风险。软件实现团体的代表应该识别与物理架构候选方案相关的软件实现的挑战、约束、可行性和风险。

3）敲定软件实现计划。应该敲定软件实现计划以反映实现软件物理架构需要的工作量。软件实现相关的工作包、任务描述和资源分配必须符合软件实现计划中确认的任务。它包括以下工作：

- 设计、编码和测试每个结构单元。
- 集成和测试结构组件。
- 设计、开发和评估原型来细化软件实现概念。
- 装配、集成和测试完整集成的软件产品配置项。

18.2.4　计算环境准备（详细架构定义阶段）

1）参与 SWE-IPT。计算环境团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的物理架构决策。来自计算环境团队的代表将他们关于计算机硬件、网络、通信、操作系统、中间件和软件架构及其他运行挑战的知识带给 SWE-IPT。

2）识别计算环境实现的挑战、约束、可行性和风险。计算环境实现团队的代表应该识别与物理架构候选方案相关的计算环境的挑战、约束、可行性和风险。

3）敲定计算环境实现计划。应该敲定计算环境实现计划以反映实现支持软件验收测试的计算环境需要的工作量。这应该包括设施准备、设备采购、安装和检查、工作站、软件应用和测试工具。与计算环境相关的工作包、任务描述和资源分配必须符合计划中确认的任务。

18.2.5　开发后的过程准备（详细架构定义阶段）

1）参与 SWE-IPT。开发后的过程实现团队的高级代表应该参与到 SWE-IPT 中以帮助
319 做出有利的物理架构决策。来自开发后的过程团队的代表将他们关于软件复制、销售、培训、支持流程的要求和挑战的知识带给 SWE-IPT。

2）识别开发后的过程实现的可行性、挑战、约束和风险。开发后的过程实现团队的代表应该识别与物理架构候选方案相关的挑战、约束、可行性和风险。

3）敲定开发后的过程实现计划。应该敲定开发后的过程实现计划以反映实现开发后的

过程需要的工作。它应该包括设施准备、设备采购、安装和检查、工作站、软件应用和支持工具。与开发后的过程相关的工作包、任务描述和资源分配必须符合计划中确认的任务。

18.2.6　软件测试和评估（详细架构定义阶段）

1）参与 SWE-IPT。软件测试和评估团队的高级代表应该参与到 SWE-IPT 中以帮助做出有利的架构决策。来自软件测试和评估的代表将他们关于测试需求和挑战带给 SWE-IPT。

2）识别软件测试和评估的可行性、挑战、约束和风险。软件测试和评估团队的代表应该识别与物理架构相关的软件测试中的挑战、约束、可行性和风险。

3）敲定软件测试计划和过程。应该敲定软件测试和评估计划及过程以反映进行软件验收测试需要的工作。必须为每个在概要架构定义中识别的软件测试用例和场景建立软件测试过程。软件测试过程应该表明在执行每个测试时应该遵循的详细步骤，包括测试准备和测试后分析活动。

4）执行软件质量保证检查和审核。应该定期进行软件质量检查以查验软件工程工件和产品来确保其完整性、准确性以及与已确定的策略和过程的一致性。实施软件质量检查是为了确保团队遵循已建立的过程和被认可的计划。在初步软件架构定义工作中应该执行以下软件质量检查：

- 检查结构配置。
- 检查结构单元规约。
- 检查结构组件规约。
- 检查软件集成和测试策略。
- 检查软件术语注册表。
- 检查需求追踪矩阵。

320

软件审核应该先于 CDR 以确保软件产品和架构工件是完整的并包含整合了已批准的变更提议和请求。应进行以下审核：

1）结构配置审核。追踪结构元素需求到其同化的功能单元。除非批准，确保集成测试不重复在较低层次的配置进行的测试。确保所有内部软件接口在软件集成策略中被恰当处理。

2）软件测试审核。追踪每个测试用例和过程到其要确认的软件需求规约。一个软件测试可能影响一个或多个软件规约（软件产品、计算环境和接口），这应该恰当地反映到需求追踪矩阵中。

3）校正行为审核。追踪每一个已授权的变更请求或提议到其校正行为配置。该评审必须确保校正行为被恰当地完成并反映在受影响的软件产品和架构工件中。

18.2.7　评审与里程碑（详细架构定义阶段）

应该以描述软件物理架构为目的来实施 CDR。CDR 旨在证实功能和物理架构是完整的、一致的，并能满足特定的软件需求。CDR 的典型议程应包括如下几点：

1. 关键设计评审——概述
   1.1. 需求基线和变更提议状态
   1.2. 功能架构状态

1.3. 物理架构状态
1.4. 关键权衡分析、结果、决策和原理
1.5. 结构单元需求可追踪性
1.6. 软件集成策略
1.7. 技术数据包状态
2. 软件实现
2.1. 最终软件实现计划
2.2. 软件实现的挑战、约束、可行性和风险
3. 计算环境
3.1. 计算环境实现计划
3.2. 计算环境评定计划
3.3. 计算环境进度表和里程碑
4. 软件测试和评估
4.1. 软件测试和评估概览
4.2. 软件测试过程
4.3. 软件质量保证
4.3.1. 软件质量检查状态
4.3.2. 软件质量审核状态
4.3.3. 软件质量进度表和里程碑
5. 软件开发后的过程
5.1. 软件开发后的过程实现计划
5.2. 软件开发后的过程评定计划
5.3. 软件开发后的过程进度表和里程碑

18.2.8　建立分配基线

应该建立分配基线以将结构单元、集成组件和结构组件规约置于技术配置的控制之下。分配基线为每个物理单元、它的接口和它到软件需求基线的可追踪性建立规约。物理单元和组件规约形成了初步软件实现活动中软件单元的设计、编码和测试的基础。软件集成策略、集成的组件、结构组件和结构组件规约构成了物理架构。应该将物理架构置于技术配置控制下，因为它为评估在软件实现期间可能提出的变更提议提供了分析基础。

321
/
322

软件实现

软件实现阶段包含将软件技术数据包转化为一个或多个经过构造、集成、测试并为验收测试做好准备的软件配置项的活动。软件实现的主要活动包括：

- 构造满足结构单元规约的软件单元。
- 组装、集成、测试软件组件使之成为一个软件配置项。
- 为具有挑战的软件组件构造原型以降低实现风险或建立概念的构造性验证。
- 执行模拟验收测试过程以保证该过程描述正确并且软件产品（包括软件配置项和计算环境）已为验收测试作好准备。

软件实现以软件构造工作开始。构造是一种生产某些东西的行为。软件构造涉及每个软件单元的程序设计、源代码编辑或编程以及测试。这一系列的技术任务表示了软件程序、子程序、模块、对象或者是图形化模型的产生方式。每个软件单元都被假定适合其预定目标或在总体架构上下文中的角色。软件构造的结果应当是一个已经依据其结构单元说明进行过测试的源代码的文档化单元。该源代码（软件单元）可以进行组装、集成以及和其他已有软件单元一起进行编译生成更大更复杂的软件组件。软件集成活动一直持续直到一个完整的、集成的、经过测试的软件配置项被实现并可用于验收测试。软件实现阶段如图 19-1 所示。

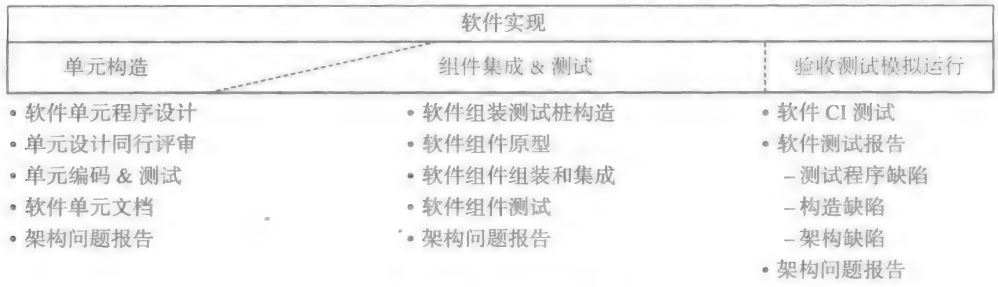


图 19-1　软件实现阶段

软件实现阶段包含了软件验收测试的模拟运行。这种模拟运行活动旨在确保验收测试过程是有效的，同时确保软件产品的运行与软件规约相符。模拟运行提供了软件产品为验收测试准备就绪的一种展示。验收测试向涉众提供了正式的演示，表明软件开发工作已经达到了它的目标。如果测试成功了，那么就将软件产品视为已经准备好交付和后期维护。宣称验收测试成功这一行为是从软件开发项目转换到部署和开发后期运作的第一步。可能会启动额外的项目来提供软件开发后的过程操作以及通过增量式或螺旋式的方法提供预先计划的产品提升。或者，软件开发项目可以简单地过渡为软件生命周期的后期开发阶段。

在软件实现阶段发现的缺陷必须被解决。那些确认是架构瑕疵导致的缺陷必须记录在架构问题报告中。SWE-IPT 必须为解决架构缺陷并修复架构工件负责。其他缺陷可能会被发现是源于程序设计或编码错误。这些软件问题报告是软件实现团队应当负责解决的。软件实

现团队不应该偏离软件技术开发计划（TDP）。而软件架构缺陷必须由 SWE-IPT 来解决。架构问题报告应由软件实现人员生成以记录已知的源于软件 TDP 的软件缺陷。

软件验收测试模拟运行的目的是确保软件产品在特定的计算环境中运行时能通过验收测
324 试。在模拟运行测试成功完成时，所有的软件问题报告的解决和回归测试、软件测试就绪评审（TRR）都应当被执行了。TRR 旨在向程序管理方和涉众代表表明软件开发已经完成并且准备好进行验收测试了。

如果程序选择绕开模拟运行测试，那么就会存在这样的风险：如果验收测试中发现任何缺陷，那么就需要进行大量的回归测试。执行模拟运行测试可确保软件产品成功通过验收测试而且在部署前不需要对软件作进一步的修改。

19.1　软件实现的产品

在这一节中定义的产品不代表在软件实现中产生的完整产品集。下列工件定义了与软件工程工作相关的软件实现工件：

1）*软件单元程序设计图*。程序设计图为每个软件单元提供计算逻辑的图形表示。它们应当描述将输入转化为预期输出所需的控制、数据以及过程流。重点应该放在数据项的算法计算上，这样做的结果是精确的数据值、错误处理机制和程序调用或与其他软件元素的接口。每个软件单元都必须进行评估以确保它满足其包括性能和其他非功能需求（例如大小、资源利用率）的规约。

2）*程序设计描述*。程序设计描述提供了一个对软件单元如何完成它的数据处理职责的叙述性、过程性的解释。

3）*软件单元测试场景*。单元测试场景应当为每个单元测试用例定义数据集、过程和预期输出。单元测试用例描述了软件单元如何进行验证以确认它满足其结构单元规约。应当开发单元测试场景并将它记录在软件开发文件夹（SDF）中。

4）*软件单元源代码文件*。源代码文件代表了计算语言相关的数据声明和指令，它们组成了软件模块、子程序、程序或者类。源代码文件可以被编译为可执行的二进制文件，该二进制文件可以运行在目标计算机系统中。

5）*软件单元测试结果*。软件单元测试的结果应当用文档存储并且能够在单元 SDF 中获取。

6）*软件组件组装、集成和测试过程*。软件集成策略应当通过建立对源代码文件进行编译、组装和链接成为一个可执行文件所需的过程来详细说明，从而支持软件组件测试。软件
325 组件测试场景和过程应该在组件 SDF 中进行描述和记录。

7）*软件集成测试结果*。软件集成测试的结果应当在组件 SDF 中进行描述和记录。

8）*模拟运行测试报告*。模拟运行测试报告应当总结测试结果、问题和遇到的错误。每一个问题或是错误应当被分配到负责解决这个问题的组织部门。必须确定需要在修复后软件产品上进行的相关回归测试的集合。这个报告将用来判断软件产品是否已准备就绪能够开始验收测试。

9）*验收测试报告*。验收测试报告应当总结测试结果、问题和遇到的错误（应当不会遇到问题或者错误！）。每一个问题或是错误应当被分配到负责解决这个问题的组织部门。必须确定需要在修复后软件产品上进行的相关回归测试的集合。这个报告用来判断软件产品是否已准备就绪能够开始运作部署。

10）软件构建过程。应当定义软件构建过程，它确立了组装、集成和验证源代码文件以生成可供发布的可执行文件的方式。构建过程应当包括以下的任务和对编译器和链接编辑器的脚本调用的自动化支持：

- 将源代码编译为二进制代码。
- 将二进制代码库包装为可提取的文件。
- 验证构建完整性。
- 分发可提取可执行映像。
- 创建文档以及/或者发布说明。
- 发布和补丁（二进制文件）协调。

11）软件问题报告。应当为在软件实现阶段发现的问题或缺陷生成软件问题报告。

12）工程变更请求（ECR）。ECR应当用来记录想要对软件架构进行的变更。每一个ECR都应当包括需要的规约和文档变更页，如果批准，这被用来融入变更并形成与提出的变更一致的架构工件和文档。

13）准备舍弃和偏离文档（如有必要）。对于基线化的软件规约中的需求的舍弃或者偏离应当作好准备并提交给项目级的变更控制委员会（CCB）来批准。偏离并没有将程序从完成某个特定需求中解放，而是允许软件产品先进行最初发布，而这个问题在将来的补丁或是版本中将会修正。舍弃则将程序从完成某个特定需求中彻底解放。 326

19.2 软件工程任务（软件实现阶段）

1）监控软件实现进程。SWE-IPT对软件实现工作进行监控以确保实现是按照物理单元所规定的那样去完成的。当提交软件工程提案时，SWE-IPT必须执行软件工程过程来确定合适的校正动作。SWE-IPT可以拒绝一个ECP提案，保持项目不偏离既定的需求基线以及功能和物理架构。

2）改进软件架构。SWE-IPT应当对软件架构进行修改以处理那些成功完成软件实现工作所需响应的变更请求。功能和物理架构可能需要进行修改和更新以反映修复某些架构缺陷所需的变更。必要时，相关支持的工件、模型以及正式文档都应当更新以反映软件架构中的变更。

3）评估ECR。SWE-IPT应该评估每一份ECR以确定期望的变更能否在项目成本和进度约束内实现。必须确定变更对整个软件架构的影响以及实现这个变更所需的工作量级别。如果需要修改功能或分配基线，SWE-IPT就应该准备工程变更提议以确定变更的范围、变更对实现项目目标和客户满意的影响以及实现变更提议所需的资源。

19.3 软件实现任务（软件实现阶段）

1）准备软件单元程序设计。软件实现团队利用选取的程序语言结构来开发流程图或者是对每个单元数据处理流程的操作原理[⊖]的其他形式描述。这些设计描述应当使用编程语言的结构化约束，最后它们都应当是人们可理解的、可验证的。

2）实施软件单元设计评审。每个软件单元在转换到编码活动之前都应当进行评审。软件实现团队中的高级代表必须参与到软件单元设计的评审中以确保软件设计与结构单元规约

⊖ 参见 *http://en.wikipedia.org/wiki/Pseudocode*

327 一致并遵从软件设计和编码标准。此外，应该对每个单元设计进行评估以确定它是否提供了相对简单、文档完备且有助于开发后期支持的方案。这些设计走查或同行评审可以只针对单个单元也可以针对相关的软件单元组。

3）准备软件单元源代码文件。软件实现团队应当根据单元程序设计文档来编辑源代码。源代码编辑器具有自动检查指令语法错误的能力，也可能有解释执行和调试的功能，所以应当使用源代码编辑器来辅助代码生成。

4）准备软件问题报告。不论何时一个软件单元测试程序发现一个错误都应当生成软件问题报告。必须对软件问题报告进行评估来确定恰当的解决方案并对它们进行追踪以确保它们被解决。只能通过调整软件架构来纠正的软件缺陷应作为 ECR 提交给 SWE-IPT。

5）准备软件单元测试场景。应该开发软件单元测试场景和测试过程并记录在软件开发文件夹中。

6）编码软件单元。软件实现团队根据单元程序设计文档为每个软件单元生成代码。

7）测试软件单元。软件实现团队对每个软件单元进行测试，以确保它的执行符合预期并且达到软件单元规约的要求。测试结果应记录在软件单元测试报告里。如果可能的话，应该通过适当地修改程序设计和修改源代码处理单元测试时发现的缺陷。必须生成软件问题报告以记录识别出的源于结构单元规约的缺陷。

8）准备软件单元测试报告。软件单元测试的结果应记录在软件单元测试报告和软件单元 SDF 中。软件单元测试报告应该确定在测试过程中遇到的每一个错误，并确定记录缺陷的软件问题报告。

9）处理发现的缺陷。如果软件单元测试发现了缺陷，那么软件单元必须重新设计并且更新代码以反映纠正措施。需要对软件开发文件夹进行更新以保存测试结果并适当地反映“实际实现的”软件单元的设计。应该对修改后的软件单元再次进行测试以确保修改确实解决了问题报告中的问题而且没有引入任何额外的问题。

10）将软件单元置于配置控制下。一旦软件单元满足了其测试过程，代码就应该存储在软件库中并置于配置控制下。

11）准备好软件组件集成和测试过程。应该准备好软件组件集成和测试程序并记录在组
328 件 SDF 中。软件组件的测试过程不应重复与每个软件单元相关联的测试。组件测试过程旨在确保软件单元和组件的接口不会因为集成而破坏。此外，对涉及与外部元件的接口的软件组件需要进行测试，以验证该接口。

12）集成并测试软件组件。软件实现团队应该集成并测试每个软件组件以确保其执行符合预期。组件集成测试不应该重复软件单元测试，而应该专注于确保软件接口正常工作，以及软件元素集成没有引入新的问题。

13）支持模拟运行测试。软件实现团队的代表应参加验收测试的模拟运行。关于测试成功的解释可能存在一些混乱。来自计算环境、测试和评估以及实现组织的代表必须通力合作来隔离每个不成功测试的来源。应准备软件问题报告以确定每个测试失败的原因，并推荐首选的校正行为。

14）支持 ECR 评估。SWE-IPT 的软件实现代表应该参与评估 ECR 并确定恰当的架构解决方案。

19.4 计算环境任务（软件实现阶段）

1）定义并实现计算环境。应及时地设计和实现计算环境以支持软件验收测试。

2）验证计算环境。应对运行计算环境进行验证以确保计算环境满足计算环境规约。应该度量计算环境的性能以建立验证软件产品性能规约所需的集成基准。

3）支持 ECR 评估。SWE-IPT 的计算环境代表应参与评估 ECR 中并确定适当的架构解决方案。

19.5 开发后的过程任务（软件实现阶段）

1）定义并实施软件复制过程。软件复制过程应该定义在预期分发媒介上生产可执行文件的电子副本所需的设备和应用程序。应该对软件复制的过程和程序进行定义和验证以确保 329
复制过程已准备好支持软件部署。这应该包括发布媒介、说明书、保修信息的打包，这样可方便的分发至客户或零售 / 转售渠道。

2）定义并实现软件发布过程。应该对打包、发布或部署软件产品所需的软件发布过程、设备和软件应用进行定义。软件发布可能涉及：1）创建一个基于互联网的发布方法，2）雇用专业的销售团队，或者 3）为打包的产品建立发布和销售渠道。这可能包括单品发布方法、大量发送到零售或经销网点和互联网文件下载机制。应该探索国际分销方式并调查与软件产品发布相关的国家独特法规的遵从性。如果软件产品是为单一客户开发的，那么发布就涉及软件产品的交付、安装和检查。

3）定义并实现软件培训过程。必须定义并准备软件培训材料和机制。培训可能是网络教学的形式、基于软件的教程或者课堂实践培训。应该基于已建立的软件配置和已认可的工程变更提议来准备培训资料。

4）定义并实现软件维护过程。应该定义软件和客户支持过程并确定提供产品和客户支持所需的设备和软件应用。客户支持过程可能涉及电话或在线帮助平台或其他机制从而为客户提供援助并记录和追踪软件问题报告。软件维护过程还涉及帮助处理源代码文件错误以及发布软件补丁或服务包的软件开发设备和应用。

注：软件维护过程不涉及提供预先计划的产品改进或以迭代或螺旋的方式提高软件版本等工作。螺旋或增量的工作涉及一个有计划的保证组织承诺和项目实例化的软件开发工作。

5）支持 ECR 评估。SWE-IPT 的开发后的过程代表应该参与评估 ECR 和确定适当的架构解决方案。

19.6 软件测试和评估任务（软件实现阶段）

1）准备软件测试环境。必须准备好用于支持软件验收测试的软件测试环境。可能需要获取或开发特殊的设备、测试应用程序和度量数据收集和分析工具以提供负载、压力、可伸 330
缩性测试、软件产品性能基准测试和回归测试。

2）敲定软件验收测试过程。应该在软件实现期间及时敲定可支持软件产品验收测试的模拟运行的软件验收测试过程。

3）执行模拟测试。软件测试和评估团队应该在软件产品的可执行文件上执行验收测试程序。执行模拟测试的动机包括：

- 为获得经验进行验收测试程序。

- 识别和纠正测试程序的缺陷。
- 确保计算环境被正确配置以支持测试。
- 确保软件产品将圆满通过验收测试障碍。

SWE-IPT 的成员应该监控每个测试的执行以确保测试程序被遵从并且结果被准确地捕获和记录。

4）*产生软件问题报告*。如果一个软件测试过程不能产生预期的效果，那么应生成软件问题报告以找出问题并解释它是如何偏离预期效果的。需要修改软件架构或 TDP 的软件问题报告应认定为 ECR。ECR 必须由 SWE-IPT 处理，以确定解决架构设计问题的正确的变更得以实施。

5）*修改软件测试过程*。应该修改软件测试过程以纠正在模拟运行中发现的错误。因为软件测试过程定义不当或其预期结果假设错误，可能发生测试失败的情况。

6）*支持 ECR 评估*。SWE-IPT 的测试和评估代表应该参与到 ECR 评估中并确定适当的架构解决方案。

7）*进行软件质量保证检查和审核*。应该在软件实现阶段定期进行软件质量检查以评估软件是否遵从认可的工作准则与规程。应该进行以下检查：

- 检查变更请求和提议解决方案的同化。
- 检查软件问题报告解决方案。
- 检查软件单元和组件对应的软件开发文件夹。
- 检查软件集成和测试记录。
- 检查模拟测试记录。

软件审核应该在 TRR 之前进行以确保软件文档提供了一个提供开发后期支持或增量 / 演化式发展的一致的、可追踪的框架。必须对软件文档和工件进行审核以确保它们反映了
331 “实际测试”的软件产品配置。应进行下列审核：

a. *软件开发文件夹审核*。该审核依据物理单元规约评估软件单元详细设计以确保所有分配的需求已经由软件单元的设计进行处理。软件开发文件夹的审核必须确保软件单元程序设计和源代码遵守既定的软件设计和编码标准。

b. *验收测试审核*。该审核评估每个测试用例和程序到其要确认的软件规约需求的可追踪性。软件测试可能影响多个需求规约，这应正确体现在需求追踪矩阵中。

c. *计算环境准备情况审核*。该审核评估计算环境对参加软件模拟运行测试的准备情况。计算环境是软件架构的一个重要元素。如果该计算环境是不可用的，那么软件产品就不能正确地进行测试。

d. *校正行为审核*。该审核追踪每个被批准的软件工程需求到已批准的架构整改计划。校正行为审核必须确保校正行为被恰当地融入了软件架构并被反映在技术数据包中受影响的文档中。由此产生的软件实现文件必须进行检查，以确保架构校正行为被恰当地纳入其中。

19.7　评审与里程碑（软件实现阶段）

1）*软件单元设计走查*。每个软件单位都应接受同行评审，以确保其程序设计满足单元规约以及设计文件已按文件标准做好准备。一旦成功完成软件单元设计走查，该软件单元就可进行软件编码和测试工作。

2）*单元代码走查*。软件实现团队为每个软件单元实施代码走查。代码走查旨在确保该

单元的程序编码已经完成。单元代码和测试结果应当进行评审，以确保它们正确地实现了规约，源代码文档正确地反映了“实际实现”的代码，并且代码中没有木马[⊖]或后门程序[⊜]的存在。

332

3）进行测试准备评审。应该进行测试准备评审以确定软件产品和计算环境对进入正式的验收测试的准备情况。测试准备评审的典型议程应包括以下主题：

1. 测试准备评审概述
   - 1.1. 测试准备评审的目标
   - 1.2. 评审先决条件
   - 1.3. 预期结果
2. 软件准备状态
   - 2.1. 软件工程状态
     - 2.1.1. 软件架构稳定性
     - 2.1.2. 重要的工程变更请求和提议
   - 2.2. 软件实现状态
     - 2.2.1. 验收测试模拟运行结果
     - 2.2.2. 重要的软件问题报告
     - 2.2.3. 软件开发文件夹审核状态
   - 2.3. 计算环境状态
     - 2.3.1. 计算环境配置
     - 2.3.2. 计算环境准备状态
     - 2.3.3. 计算环境准备审核状态
3. 验收测试状态
   - 3.1. 测试计划概述
   - 3.2. 测试覆盖
   - 3.3. 测试进度
   - 3.4. 重要的软件问题报告
   - 3.5. 回归测试方法
4. 小结
   - 4.1. 行动项目和任务
   - 4.2. 总结
   - 4.3. 最终评语

333～334

⊖ 木马指希腊传说中的特洛伊木马的故事。它是一种伪装成正常应用的恶意程序。详情请见：http://netsecurity.about.com/cs/generalsecurity/g/def_trojan.htm

⊜ 后门程序是一种秘密的或无痕迹的侵入计算机系统的方式。详情请见：http://netsecurity.about.com/cs/generalsecurity/g/def_backdoor.htm

第 20 章

Software Engineering: Architecture-Driven Software Development

软件验收测试

验收测试是一种正式的测试活动，其参与者包括企业、客户和涉众代表，从而见证软件产品已为部署作好准备。如果为软件开发项目建立了合同，那么该活动就代表一个表明该项目已经完成合同义务的重要步骤。如果该项目是由企业内部的资源资助开发，那么这个活动就提供了项目的需求已经得到满足且该产品已为部署作好准备的证据。这些产品可能分布在内部业务流程中，也可能以消费者软件包的形式出售。

在软件部署之前，软件配置项必需接受最后的检测，以保证软件数据包是完整的。必须对架构技术数据包（TDP）进行审核，以确保它能准确地反映所构建和测试的软件配置。功能配置审核 (FCA) 检查软件测试的结果，以确保软件产品满足由变更提议补充过的规约。物理配置审核 (PCA) 检查指定的软件部署数据包 (DDP)，以确保当前构建和测试的软件配置在它的文档集中准确地反映出来。这些配置评审应该用于建立软件产品配置与架构和配置 DDP 的一致性。

部署准备评审（DRR）应该提交验收测试和软件配置审核的结果以及每一个开发后的过程的状态。部署准备评审旨在确保软件的复制、销售、培训以及维护的过程已作好准备能够支持客户和涉众对软件产品培训和问题解决的需求。图 20-1 描绘了软件开发的验收测试阶段的产品。

图 20-1　验收测试阶段

20.1　软件验收测试的产品

1）验收测试报告。验收测试报告总结了验收测试的结果。它应当提供所生成的软件问题报告到其行为不满足测试程序的软件构件和单元的可追踪性。验收测试确认软件产品（软件配置项和计算环境）满足软件规约。

2）软件问题报告。软件问题报告应该包括验收测试过程中所有与规约不一致的问题。应该对优先级、风险、变通方法和可选方案进行识别，这样就能够确定恰当的解决方案从而可以很快实施软件部署。如果进行了模拟测试，那么在验收测试过程中就不应该生成任何新的软件问题报告。

3）软件的偏离与舍弃。软件的偏离和舍弃应该是由那些与软件规约相冲突的软件产品中的缺陷造成的。偏离表示软件产品和软件规约的不一致性，而舍弃表示对这些不一致情形的承认并请求批准在当前非一致状态下，进行软件产品的部署。偏离提供对部署软件产品的 336
暂时批准，同时承诺在未来的补丁和发布版本中解决问题。舍弃适用于软件不满足合同或软件需求的情形，通过舍弃需求，软件产品才得以部署，正因这样，也就不承诺在未来的补丁和发布版本中解决这些缺点。

4）明确的软件部署数据包。软件部署数据包（DDP）应该包含允许软件未来的扩展和增强的实现工件。明确的软件部署数据包应该附有一份所有已批准并融入软件架构和设计工件中的变更清单。此外，对于软件产品的每个配置项，应该有一个独立的软件部署数据包。

- 最终的软件可执行文件。最终的软件可执行文件应该为软件复制和销售确立一个基线。
- 最终的软件源文件。最终的软件源文件应该并入软件产品基线中。一个软件材料的账单（SBOM）应当记录每一份源代码文件，这些源代码文件构成最终的软件产品配置。
- 最终的构建步骤。最终的构建步骤应该被记录下来，以解释如何生成软件的执行文件。软件维护团队将会利用这些步骤生成补丁或软件产品的未来版本。
- 最终架构的技术数据包。更新的软件架构工件（需求基线、功能和物理架构、软件术语表、模型等）应该被记录下来，以提供一个后期开发维护以及软件产品和过程改进的基础。

5）明确的开发后的过程 DDP。对于每一个开发后的过程的最终 DDP，都应该作如下的安排和布局，具体包括设施、设备、工作站、通信工具、网络系统和设备连接、软件工具以及数据库等，这些安排和布局使得后期开发的每一个过程更加方便。这些 DDP 提供了详尽的图表和必要的支撑文档资料，以维护后期的每一个过程。

20.2　软件工程（软件验收测试阶段）

1）见证软件验收测试。SWE-IPT 的代表们应该见证验收测试的实施，以确保这些测试是按照软件测试过程实施的。

2）见证开发后的过程的确认。SWE-IPT 的代表们应该见证开发后的过程确认测试的实施，以确保实施的测试符合测试过程。 337

3）再评估软件架构。SWE-IPT 应该对软件架构进行再评估，以理解软件问题的影响并评估潜在的解决方案。在这个软件开发活动的较晚时间节点，缺陷的解决也许不太可能。因此，SWE-IPT 应该准备软件的偏离和舍弃，并适当地，针对悬而未决的问题给出处理对策。

4）支持软件配置评审。SWE-IPT 的代表们应该参加软件功能配置审核（FCA）和物理配置审核（PCA）的实施。

5）准备最终的架构技术数据包。SWE-IPT 应该准备好更新的架构工件，以支持软件配置的审核。明确的架构数据包应该附有一份所有已批准并融入软件架构和文档设计工件（文档、图表和模型等）中的变更清单。

20.3　软件实现组织（软件验收测试阶段）

1）监督软件验收测试。软件实现组织的代表们应该见证验收测试的实施，以了解软件行为以及对测试场景和条件的反应。

2）评估软件问题报告。软件实现组织必须确定解决在验收测试过程中所出现问题的纠错行动。由验收测试结果生成的软件问题报告应该接受评估，以决定这些问题是不是由软件实现缺陷造成的。如果有必要，可以提高问题的等级，由 SWE-IPT 通过修改软件架构或寻求偏差或舍弃来解决。

3）提供最终的软件可执行文件。应该生成最终的软件可执行文件并交付给项目配置管理组织，以支持软件复制过程和软件配置审核。

4）交付最终的软件源文件。最终的源代码文件应该为软件问题的隔离和解决提供基础。源代码文件也要为软件产品基线的未来扩展和增强提供基础。

5）交付软件构建步骤。应该准备最终的软件构建步骤并交付给项目配置管理组织，以方便实施软件复制过程。软件构建步骤也被软件复制团队用来生成分发媒介、提供软件产品的补丁和未来发布的版本。

6）支持软件配置审核。软件实现组织的代表们应该参加软件功能配置审核和物理配置
338 审核的实施。

20.4　计算环境实现组织（软件验收测试阶段）

1）支持软件验收测试。计算环境组织的代表应该参与到验收测试的实施中，以了解计算环境实现的行为。在验收测试过程中，开发团队的成员可能会对需求的解释或行为规约产生疑惑。计算环境是软件产品要素之一，也可被软件验收测试所确认。

2）交付计算环境的软件部署数据包（DDP）。最终的计算环境 DDP 应该用于支持软件配置审核。最终的计算环境 DDP 应该包括计算环境物理特征和性能基准的详细规约。

3）支持软件配置审核。计算环境组织的代表应该参与软件功能配置审核和物理配置审核。

20.5　开发后的过程组织（软件验收测试阶段）

1）敲定数据处理流程。为了执行软件部署或维护任务，开发后的过程组织应当敲定数据处理流程。流程应当建立一些如何开展典型的任务和处理非典型的情况的指导原则。

2）确认开发后的过程。开发后的过程组织应当为软件的复制、销售、培训以及支持过程准备测试过程。应该确认每一个开发后的过程都为软件维护过程的实施做好准备。

3）敲定开发后的过程的软件部署数据包（DDP）。应该准备好每一个开发后的过程的最终部署数据包以提供维护每一个过程所需的详细设计图表和必要的文档。

20.6　软件测试和评估（软件验收测试阶段）

1）执行验收测试过程。软件测试和评估组织应该执行验收测试过程以确保软件配置和计算环境合格。对于任何测试失败都要分析失败的原因。软件质量保证团队的成员应该监督每一个测试的执行以确保遵从测试程序以及正确地捕获或记录测试结果。 339

2）生成软件问题报告。如果一个软件测试程序没能生成期望的结果时，那么就该生成一份软件问题报告，以识别其中的问题，以及它是如何偏离期望值的。软件问题报告应该以能够再现测试结果的方式记录遇到的每个问题。每个问题报告应该分配给适合的软件开发团队解决。

3）发布验收测试报告。软件测试和评估组织应当准备验收测试报告，以记录软件验收测试的状态。这份报告应当记录所有已发现的软件缺陷并建立表明修复的软件配置或计算环境特征能充分解决软件缺陷所需的回归测试。

20.7　评审与里程碑（软件验收测试阶段）

1）功能配置评审。软件工程集成产品团队主导软件配置的评审，以确保需求已经被正确地实现、测试和满足。软件规约中的每个需求都应该追踪到测试结果，这些结果证实了软件实现的适合性。所有批准的工程变更请求（ECP）和软件问题报告都应该进行评估，以确保它们已被解决和融入软件 DDP。

2）物理配置评审。软件工程集成团队主导软件配置的评审，以确保需求被适当地实现、测试和满足。该评审应确保所有批准的 ECP 和软件问题报告已被解决。

3）部署准备评审。软件部署准备评审应该用于评价验收测试和配置评审的结果。每一个开发后的过程确认的状态都必须进行评审，以确保支持软件部署的准备工作就绪。只要部署准备评审成功完成，软件产品就可以转换到生命周期中的软件部署阶段。应该认为软件开发项目已结束，除非它采用迭代式的或螺旋式的开发方法。一个典型的部署准备评审议程应该包括下列主题：

1. 概述
   - 1.1. 议程
     - 1.1.1. 部署准备评审的目标
     - 1.1.2. 评审先决条件
     - 1.1.3. 期望的结果
   - 1.2. 项目概述（可选）
     - 1.2.1. 项目关闭准则
     - 1.2.2. 软件业务的转换
2. 软件开发状态
   - 2.1. 软件工程状态
     - 2.1.1. 最终的架构 TDP 状态
     - 2.1.2. 重要的工程变更提议
     - 2.1.3. 重要的舍弃和偏离
   - 2.2. 软件实现准备

2.2.1. 软件实现准备状态
2.2.2. 软件 DDP 状态
2.2.3. 重要的软件问题报告
2.3. 计算环境准备
2.3.1. 计算环境准备状态
2.3.2. 计算环境 DDP 状态
2.3.3. 重要的软件问题报告
2.4. 测试和评估状态
2.4.1. 验收测试结果
2.4.2. 功能配置评审状态
2.4.3. 物理配置评审状态
3. 开发后的过程准备
3.1. 软件复制过程准备
3.2. 软件销售过程准备
3.3. 软件培训过程准备
3.4. 软件维护过程准备
4. 小结
4.1. 行动项目和任务
4.2. 结论
4.3. 最终评语

20.8　建立软件产品基线

软件产品基线（SPB）应该在开发准备评审（DRR）结束时确立。为了发布软件产品的初始版本，软件产品基线将最终的架构 TDP 和软件 DDP 融合为一个完整的基线。该产品基线应该提供给软件维护组织，作为软件问题解决方案以及为软件产品的扩展和增强提供工程解决方法的基础。如果使用增量的或演化的开发方法，那么软件产品基线应该提供给下一个程序组织，该组织对软件产品的增强承担责任。

340
~
343

索　引

注：页码后面的“f”和“t”分别指图和表。
索引中的页码为英文原书页码，与书中页边标注的页码一致。

C

D

E

F

Q

R

S

T

U

V

W

推荐阅读

软件工程：实践者的研究方法（第7版）

作者：（美）Roger S. Pressman
译者：郑人杰 等
ISBN：978-7-111-33581-8
定价：79.00元

需求工程：基础、原理和技术

作者：（德）Klaus Pohl
译者：彭鑫 等
ISBN：978-7-111-38231-7
定价：89.00元

软件可靠性方法

作者：（以色列）Doron A. Peled
译者：王林章 等
ISBN：978-7-111-36553-2
定价：45.00元

需求工程：实践者之路（原书第4版）

作者：（德）Christof Ebert
译者：洪浪
ISBN：978-7-111-43986-8
定价：69.00元

软件建模与设计：UML、用例、模式和软件体系结构

作者：（美）Hassan Gomaa
译者：彭鑫 等
ISBN：978-7-111-46759-5
定价：85.00元

软件工程 架构驱动的软件开发

Software Engineering Architecture-Driven Software Development

首次针对IEEE软件工程知识体系（SWEBOK）标准中基本技能进行广泛解读

应用系统工程的原理和原则推进更好的软件开发

激励软件团队进行广泛对话，并通过对话来构建软件工程关键原理和实践的集合

本书首次针对IEEE软件工程知识体系标准中一些基本技能进行广泛解读。软件工程标准方面的专家Richard Schmidt解释了为政府或公司开发项目时哪些传统的软件工程实践会获得认可。

软件工程教育往往缺乏标准，许多教育机构更多地关注软件产品的实现而不是设计，而设计实际上影响软件产品架构。许多加入产品开发大军的毕业生由于缺乏某些必要的技能，导致一些软件项目要么彻底失败，要么超出预算并且不能按期提交。

另外，软件工程师也应该理解系统工程和架构，即程序将要运行的硬件和外围设备是如何布局的。随着更多程序需要进行并行计算，这个问题将变得更加重要，所以需要充分理解处理器和硬件的并行处理能力。本书也会帮助软件开发人员和系统工程师深入分析他们具有的技能是如何相互支持与补充的。通过聚焦这些关键的知识点，软件工程师掌握了大量最好的实践知识，可以在任何企业或领域的软件产品开发中发挥作用。

主要特色

- 涵盖解决交叉学科软件知识领域和主题的软件工程实践。
- 提供建立健壮架构的最佳实践，从而有助于生成、集成和测试代码。
- 有助于准确地估计项目开发成本和进度，从而提高软件开发的成功率。
- 揭示如何将软件工程和系统工程关联起来，促进在项目环境中与其他的工程技术人员沟通。

本书从交叉学科的角度阐述软件开发。根据IEEE计算机学会软件工程知识体系，本书重点介绍如何集成各种软件开发方法和架构设计实践，这些方法和实践对于开发行之有效的软件产品至关重要。作者对于软件产品的开发采用了传统的软件工程实践。无论你是软件开发人员、系统工程师还是项目经理，本书都颇具参考价值。

本书译自原版*Software Engineering: Architecture-Driven Software Development*并由Elsevier授权出版

ELSEVIER 爱思唯尔

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

封面设计 · 锡 林

上架指导：计算机\软件工程

ISBN 978-7-111-53314-6

9 787111 533146 >

定价：69.00元